(12) UK Patent Application (19) GB (11) 2 325 073 (13) A

(43) Date of A Publication 11.11.1998

(21) Application No 9813753.2

(22) Date of Filing 30.01.1996

Date Lodged 25.06.1998

(30) Priority Data
(31) 08382781 (32) 31.01.1995 (33) US

(62) Divided from Application No 9601870.0 under Section 15(4) of the Patents Act 1977

(51) INT CL[6]
H03M 11/04 , B41J 5/10

(52) UK CL (Edition P)
G4H HKK H13D H14A
B6F FCBK

(56) Documents Cited
GB 2283598 A GB 2266797 A US 4650927 A

(58) Field of Search
UK CL (Edition P) G4H HKH HKK
INT CL[6] G06F 3/023 , H03M 11/04 11/08 , H04M 1/00 1/272 1/274 3/62 9/00 11/06

(71) Applicant(s)
Fujitsu Limited
(Incorporated in Japan)
1-1 Kamikodanaka 4 -chome, Nakahara-ku,
Kawasaki-shi, Kanagawa 211-8588, Japan

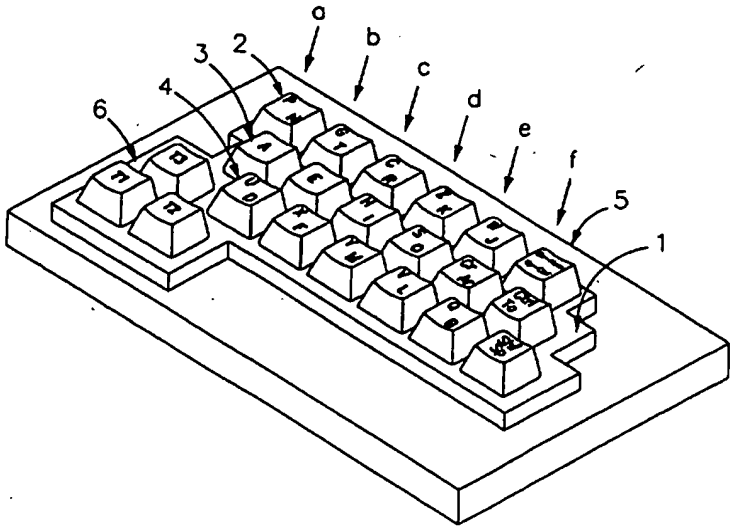
(72) Inventor(s)
Masakatsu Sugimoto

(74) Agent and/or Address for Service
Haseltine Lake & Co
Imperial House, 15-19 Kingsway, LONDON,
WC2B 6UD, United Kingdom

(54) Abstract Title
Single handed keyboard with character ambiguity resolution logic

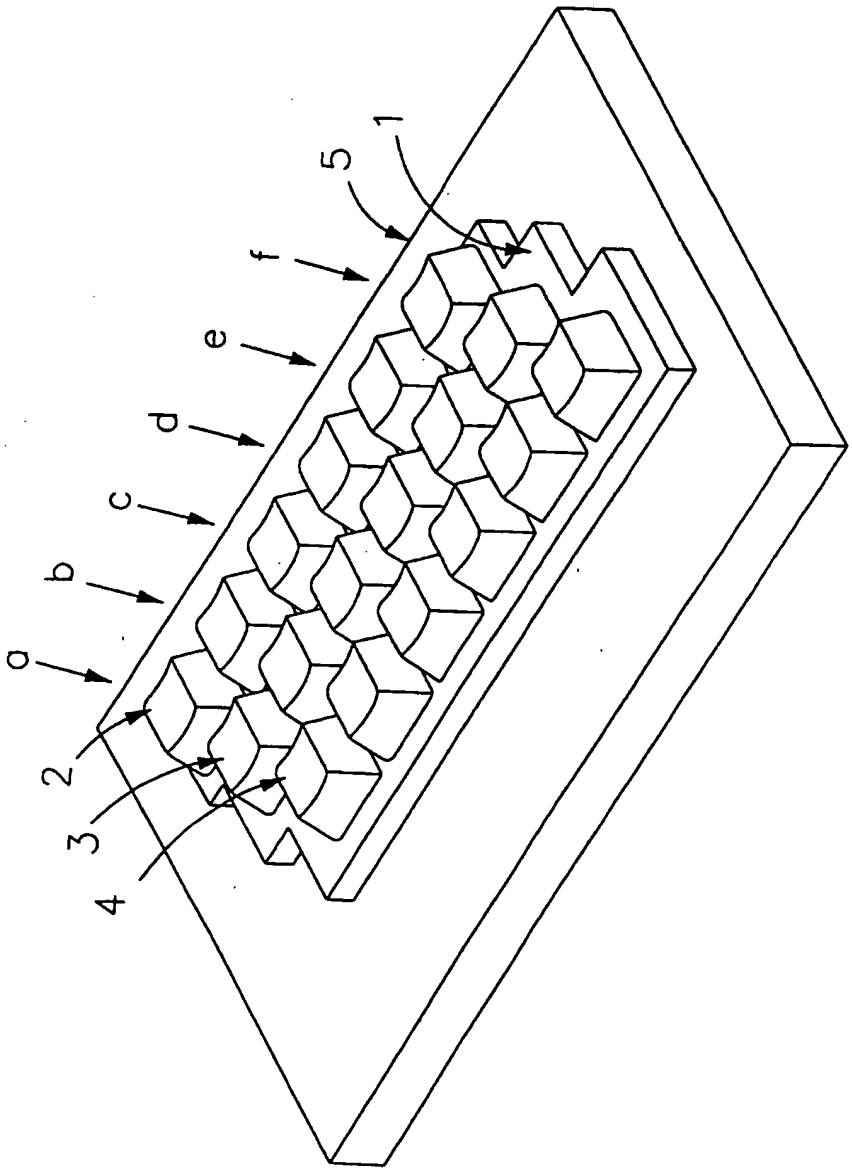
(57) The keyboard uses ambiguity resolution logic (Fig.8) to resolve the correct character for a sequence of keys with multiple characters assigned thereto chosen by a user. The user presses a key Ar to invoke the ambiguity resolution after a string of keys has been entered, such as for a complete word. This causes the resolution logic to generate from a dictionary a list of word choices corresponding to the possible spellings of words based on the key string entered. These choices are displayed to the user and if the displayed sequence of characters or word is not the desired one, the Ar key can be pushed again until the desired sequence or word appears. The system allows the user to enter a character string without having to simultaneously enter a thumb key to select the alternate character assigned to that multi character key.

FIG.2



GB 2 325 073 A

FIG. 1



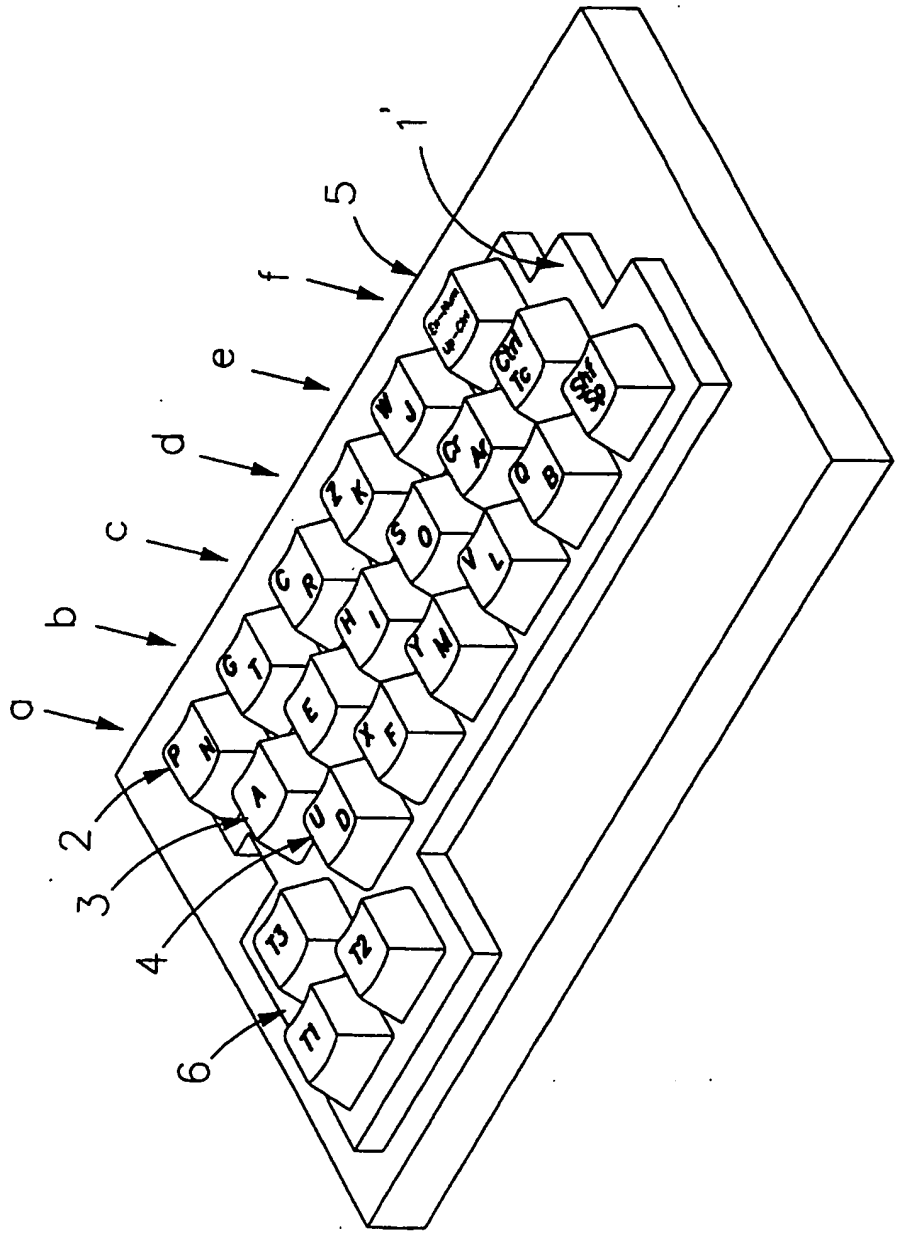


FIG. 2

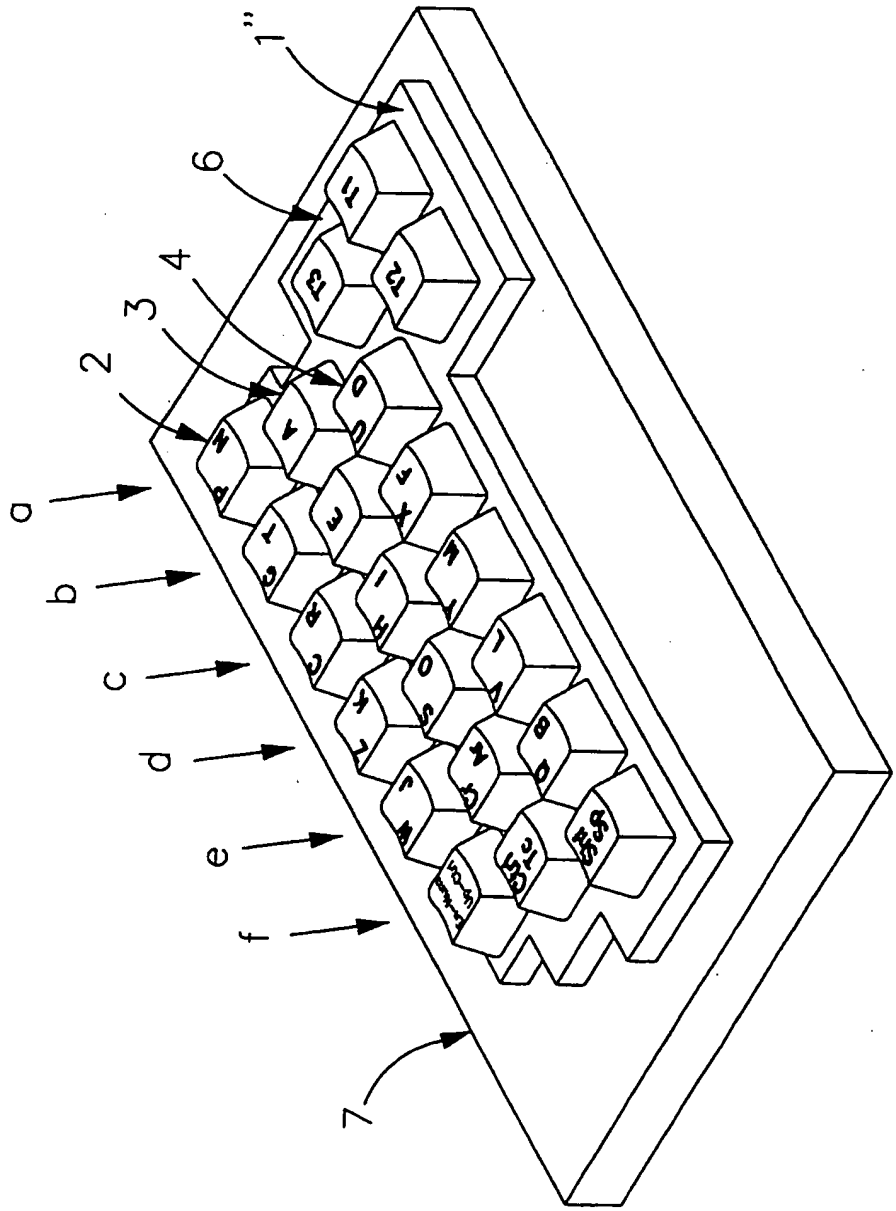
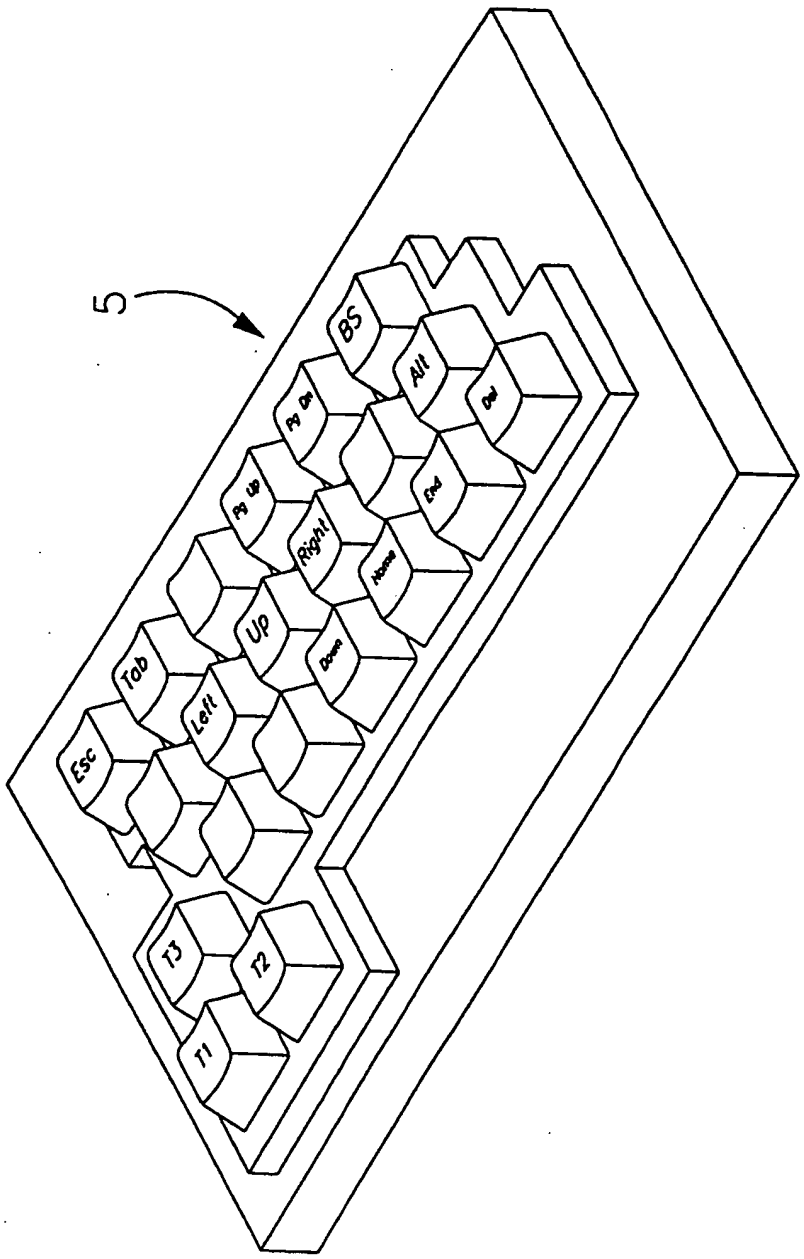


FIG.3

FIG. 4


5
BS
Alt
Del
Pg Dn
Pg Up
End
Right
Home
Up
Down
Tab
Left
Esc
T3
T2
T1

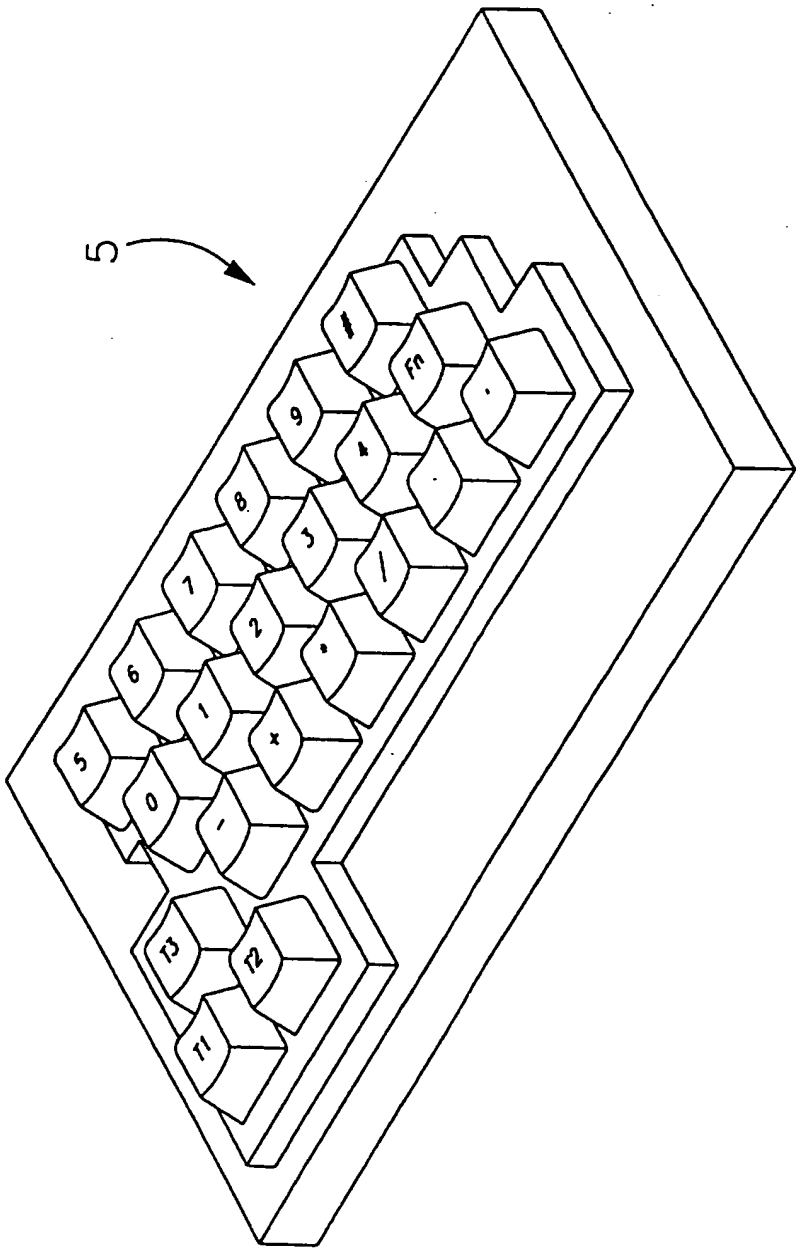
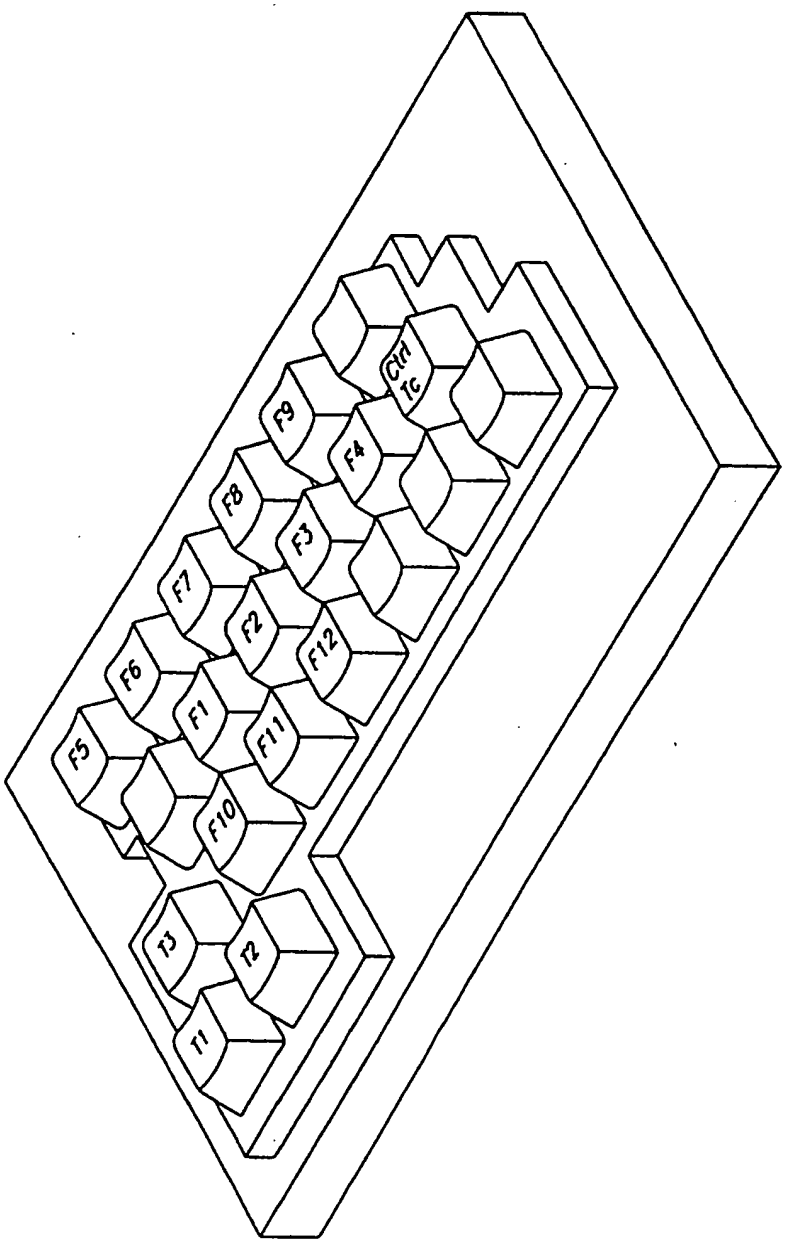


FIG. 5

FIG. 6



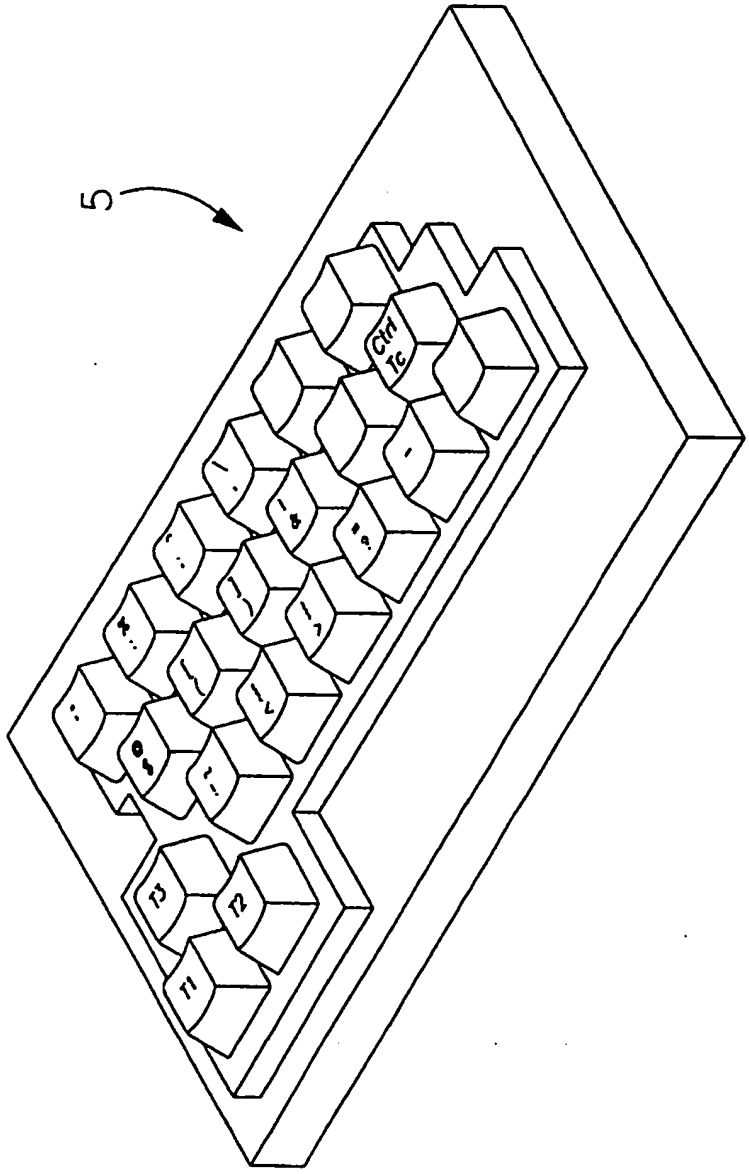
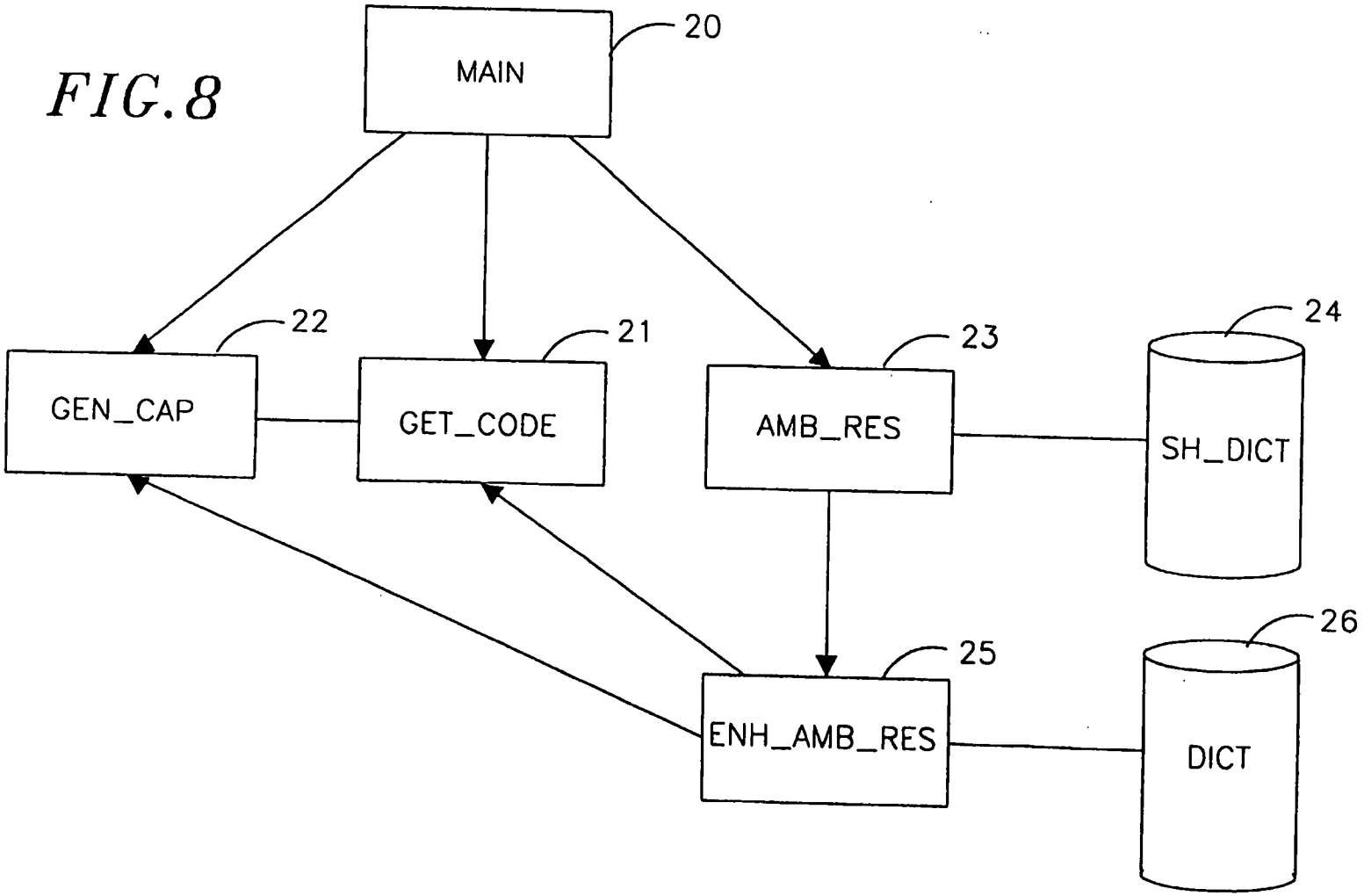


FIG. 7

FIG. 8


MAIN
20
GEN_CAP
22
GET_CODE
21
AMB_RES
23
SH_DICT
24
ENH_AMB_RES
25
DICT
26

8124

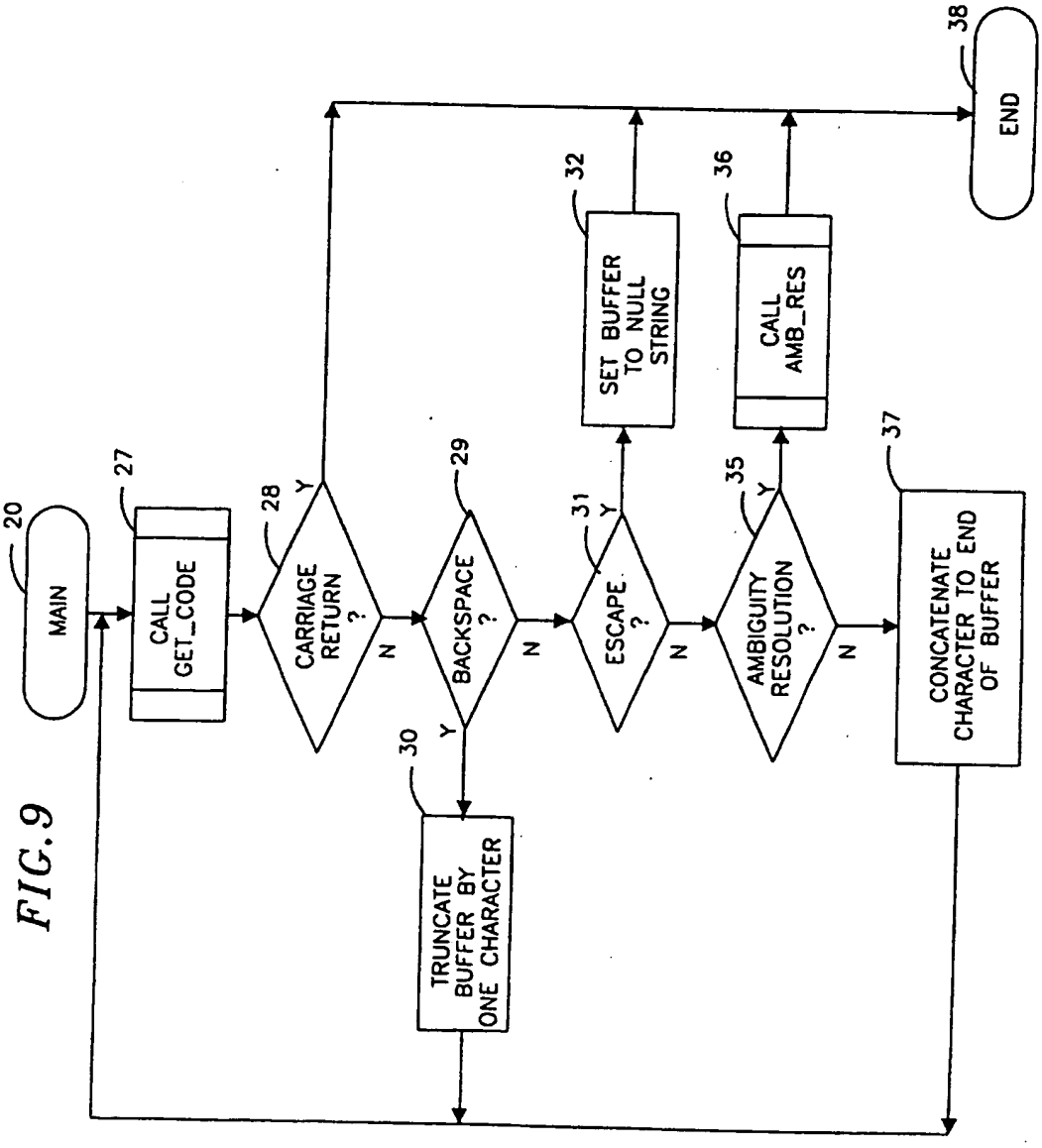

20
MAIN
27
CALL GET_CODE
28
CARRIAGE RETURN ?
Y
N
29
BACKSPACE ?
Y
N
30
TRUNCATE BUFFER BY ONE CHARACTER
31
ESCAPE ?
Y
N
32
SET BUFFER TO NULL STRING
35
AMBIGUITY RESOLUTION ?
Y
N
36
CALL AMB_RES
37
CONCATENATE CHARACTER TO END OF BUFFER
38
END

FIG. 9

FIG. 10

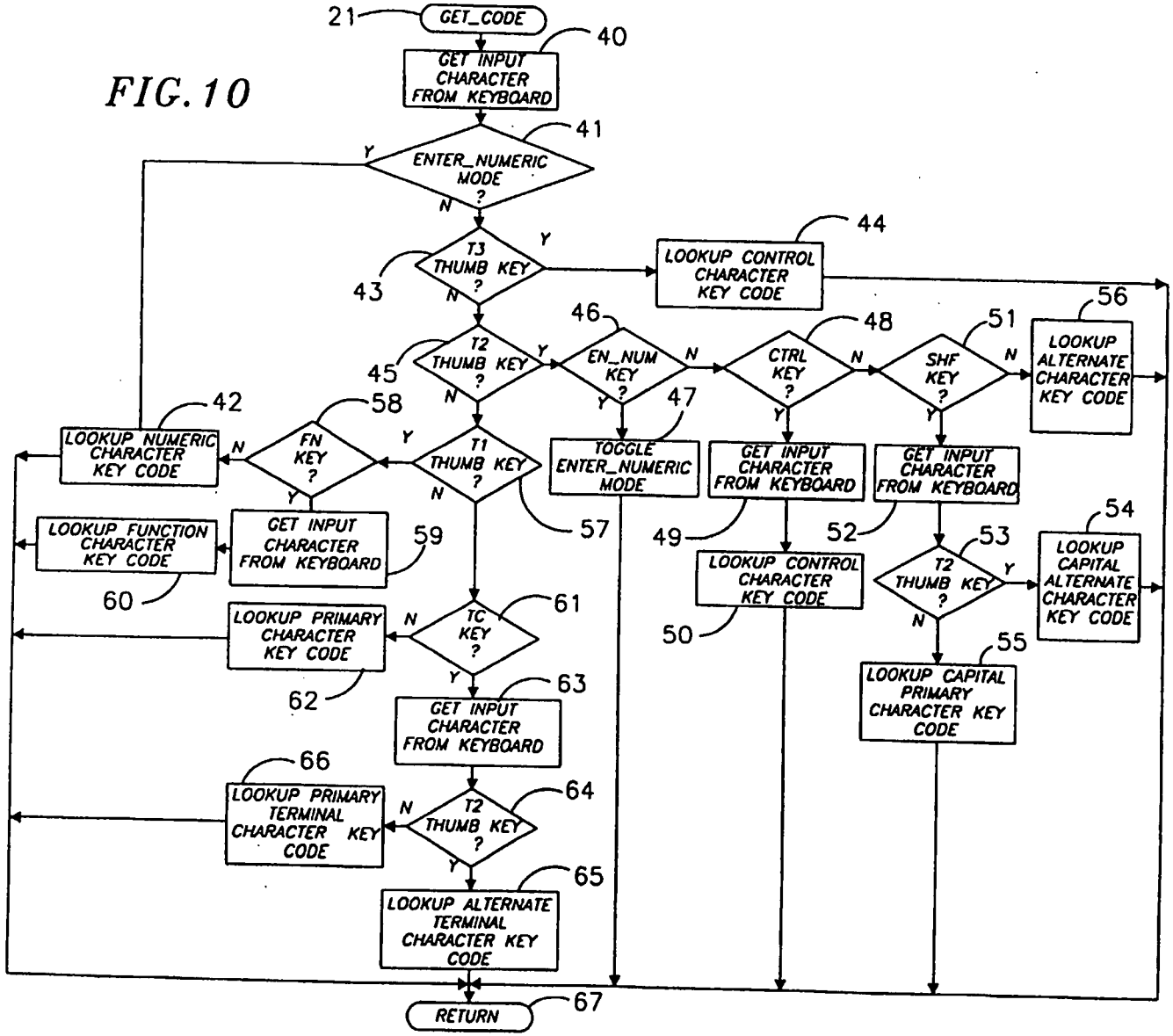

21
GET_CODE
40
GET INPUT CHARACTER FROM KEYBOARD
41
ENTER_NUMERIC MODE ?
Y
N
T3 THUMB KEY ?
43
44
LOOKUP CONTROL CHARACTER KEY CODE
T2 THUMB KEY ?
45
46
EN_NUM KEY ?
47
TOGGLE ENTER_NUMERIC MODE
CTRL KEY ?
48
GET INPUT CHARACTER FROM KEYBOARD
49
LOOKUP CONTROL CHARACTER KEY CODE
50
SHF KEY ?
51
GET INPUT CHARACTER FROM KEYBOARD
52
T2 THUMB KEY ?
53
LOOKUP CAPITAL PRIMARY CHARACTER KEY CODE
55
LOOKUP CAPITAL ALTERNATE CHARACTER KEY CODE
54
LOOKUP ALTERNATE CHARACTER KEY CODE
56
T1 THUMB KEY ?
57
FN KEY ?
58
GET INPUT CHARACTER FROM KEYBOARD
59
LOOKUP FUNCTION CHARACTER KEY CODE
60
LOOKUP NUMERIC CHARACTER KEY CODE
42
TC KEY ?
61
LOOKUP PRIMARY CHARACTER KEY CODE
62
GET INPUT CHARACTER FROM KEYBOARD
63
T2 THUMB KEY ?
64
LOOKUP PRIMARY TERMINAL CHARACTER KEY CODE
66
LOOKUP ALTERNATE TERMINAL CHARACTER KEY CODE
65
RETURN
67

FIG. 11

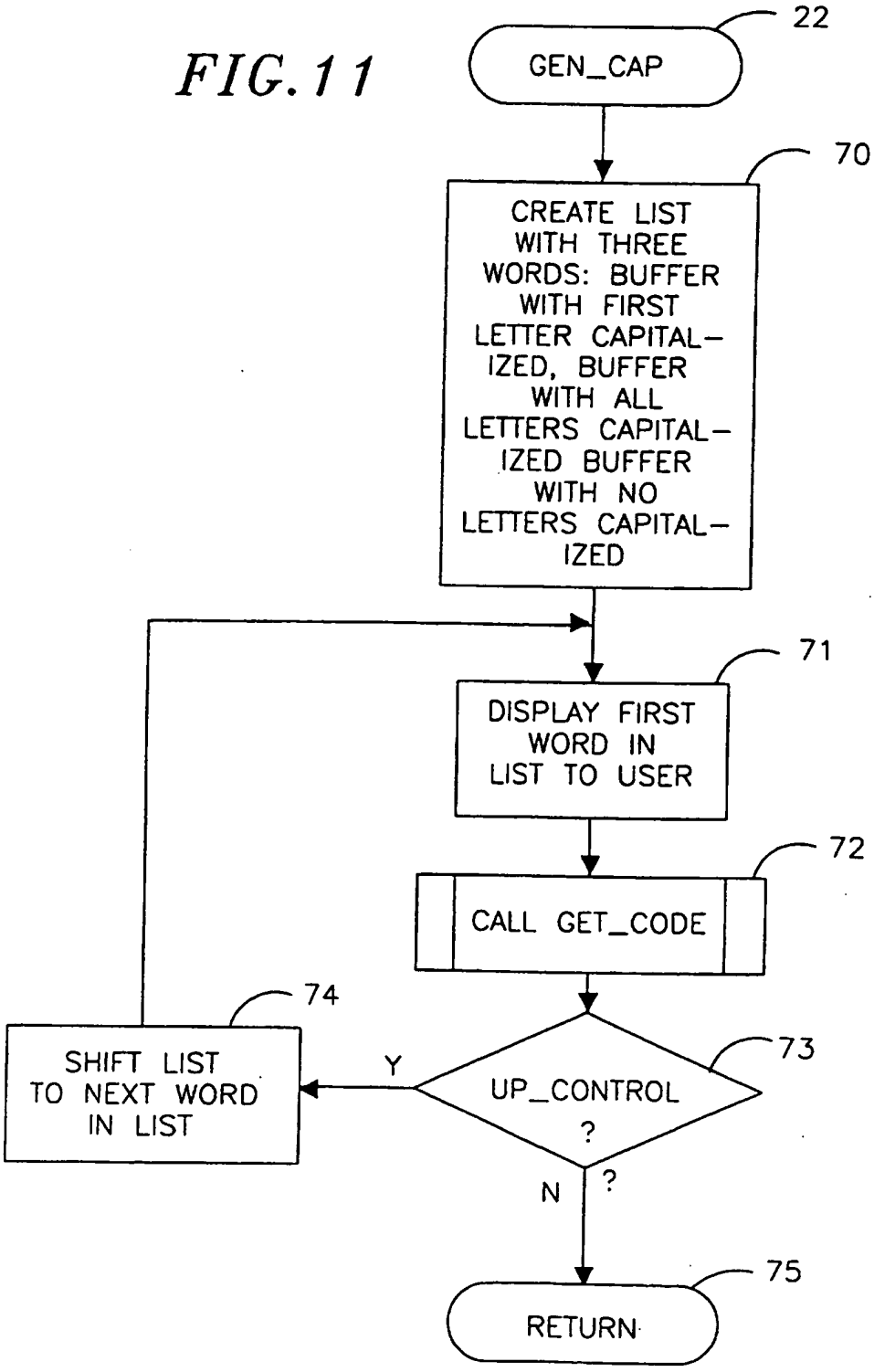

22
GEN_CAP
70
CREATE LIST WITH THREE WORDS: BUFFER WITH FIRST LETTER CAPITAL-IZED, BUFFER WITH ALL LETTERS CAPITAL-IZED BUFFER WITH NO LETTERS CAPITAL-IZED
71
DISPLAY FIRST WORD IN LIST TO USER
72
CALL GET_CODE
73
UP_CONTROL ?
Y
74
SHIFT LIST TO NEXT WORD IN LIST
N
?
75
RETURN

FIG. 12

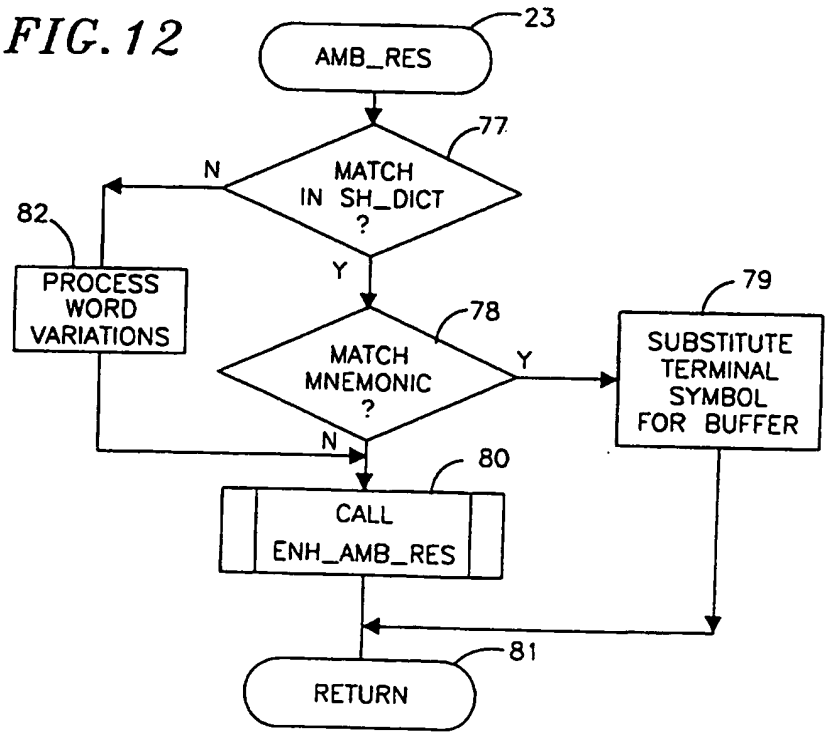

AMB_RES
23
MATCH IN SH_DICT ?
77
N
Y
PROCESS WORD VARIATIONS
82
MATCH MNEMONIC ?
78
Y
N
SUBSTITUTE TERMINAL SYMBOL FOR BUFFER
79
CALL ENH_AMB_RES
80
RETURN
81

FIG. 13

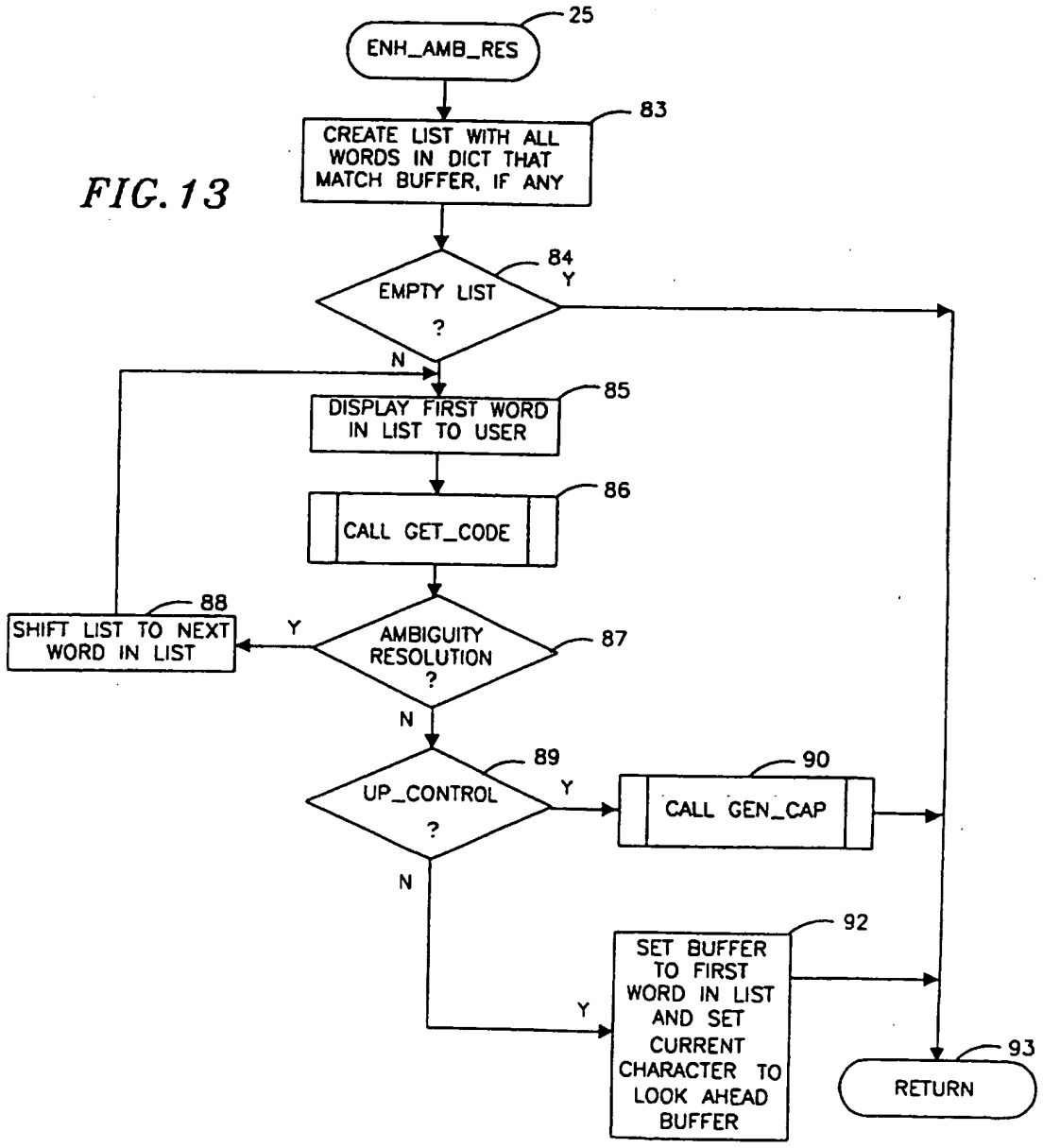

25
ENH_AMB_RES
83
CREATE LIST WITH ALL WORDS IN DICT THAT MATCH BUFFER, IF ANY
84
EMPTY LIST ?
Y
N
85
DISPLAY FIRST WORD IN LIST TO USER
86
CALL GET_CODE
88
SHIFT LIST TO NEXT WORD IN LIST
Y
AMBIGUITY RESOLUTION ?
87
N
89
UP_CONTROL ?
Y
90
CALL GEN_CAP
N
92
SET BUFFER TO FIRST WORD IN LIST AND SET CURRENT CHARACTER TO LOOK AHEAD BUFFER
Y
93
RETURN

FIG. 14

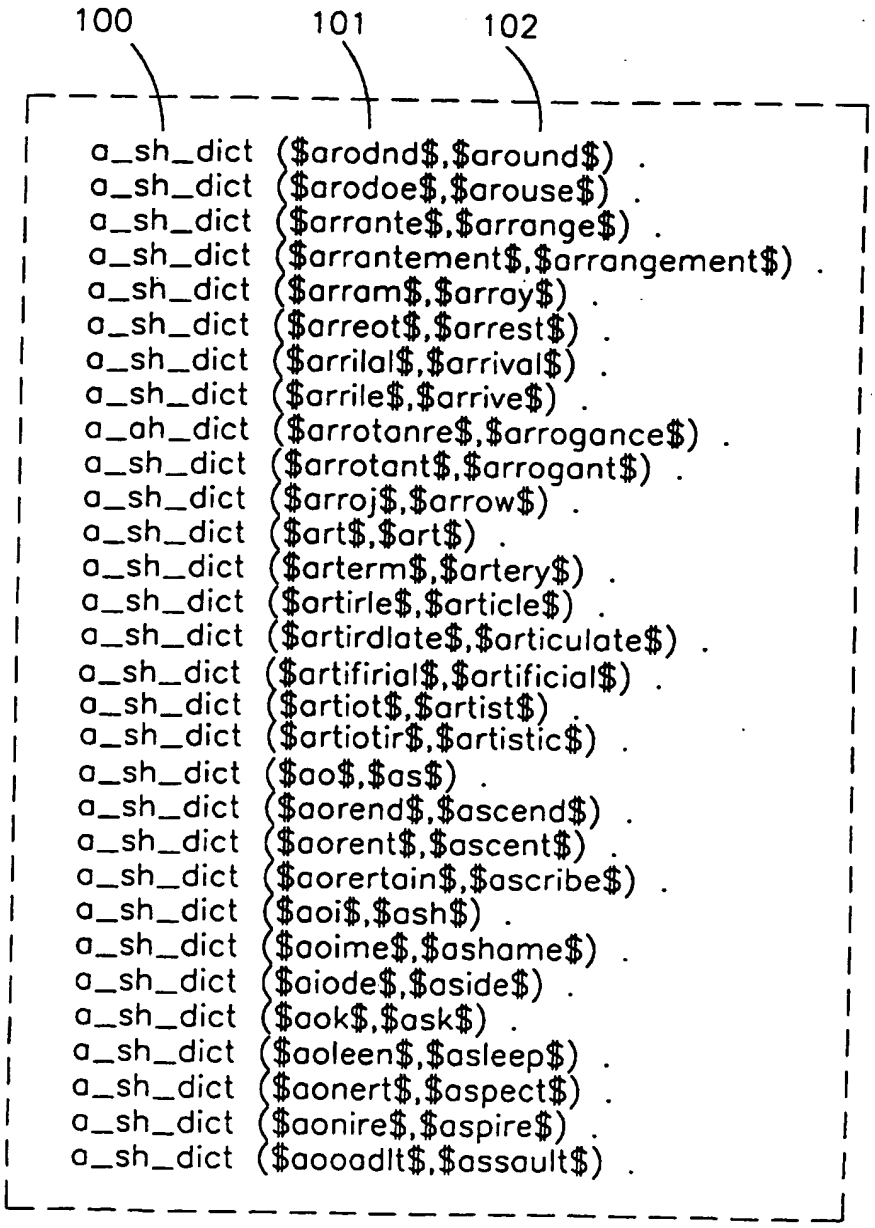

100
101
102
a_sh_dict ($arodnd$,$around$) .
a_sh_dict ($arodoe$,$arouse$) .
a_sh_dict ($arrante$,$arrange$) .
a_sh_dict ($arrantement$,$arrangement$) .
a_sh_dict ($arram$,$array$) .
a_sh_dict ($arreot$,$arrest$) .
a_sh_dict ($arrilal$,$arrival$) .
a_sh_dict ($arrile$,$arrive$) .
a_ah_dict ($arrotanre$,$arrogance$) .
a_sh_dict ($arrotant$,$arrogant$) .
a_sh_dict ($arroj$,$arrow$) .
a_sh_dict (art,art) .
a_sh_dict ($arterm$,$artery$) .
a_sh_dict ($artirle$,$article$) .
a_sh_dict ($artirdlate$,$articulate$) .
a_sh_dict ($artifirial$,$artificial$) .
a_sh_dict ($artiot$,$artist$) .
a_sh_dict ($artiotir$,$artistic$) .
a_sh_dict (ao,as) .
a_sh_dict ($aorend$,$ascend$) .
a_sh_dict ($aorent$,$ascent$) .
a_sh_dict ($aorertain$,$ascribe$) .
a_sh_dict (aoi,ash) .
a_sh_dict ($aoime$,$ashame$) .
a_sh_dict ($aiode$,$aside$) .
a_sh_dict (aok,ask) .
a_sh_dict ($aoleen$,$asleep$) .
a_sh_dict ($aonert$,$aspect$) .
a_sh_dict ($aonire$,$aspire$) .
a_sh_dict ($aoooadlt$,$assault$) .

FIG. 15

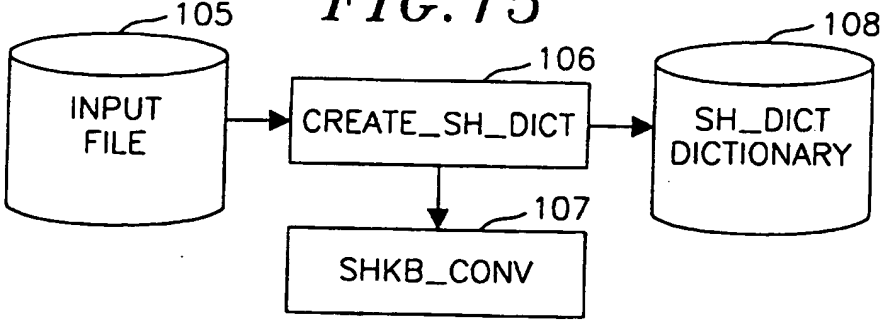

105
INPUT FILE
106
CREATE_SH_DICT
108
SH_DICT DICTIONARY
107
SHKB_CONV

FIG. 16

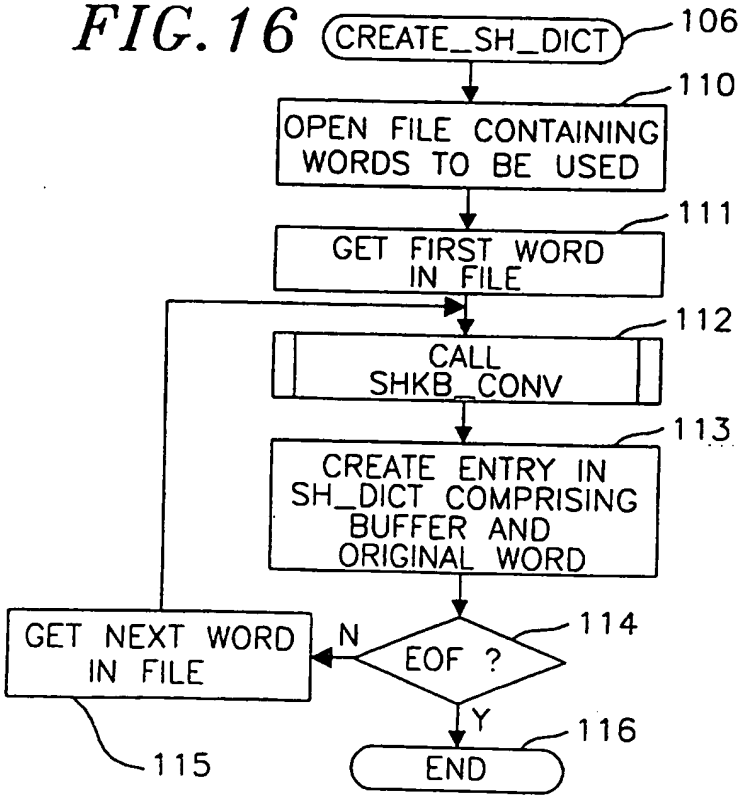
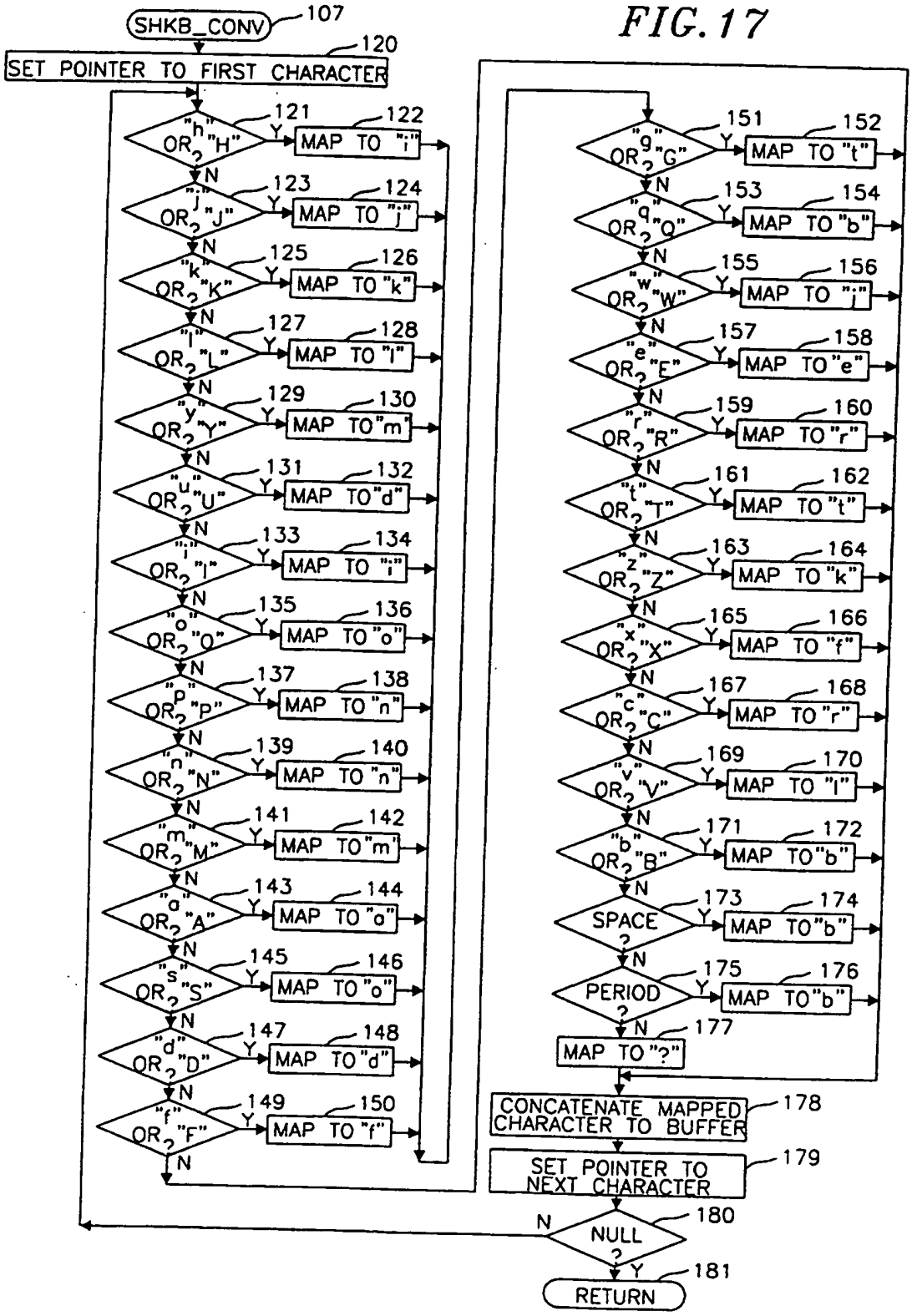

CREATE_SH_DICT
106
OPEN FILE CONTAINING WORDS TO BE USED
110
GET FIRST WORD IN FILE
111
CALL SHKB_CONV
112
CREATE ENTRY IN SH_DICT COMPRISING BUFFER AND ORIGINAL WORD
113
EOF ?
114
N
Y
GET NEXT WORD IN FILE
115
END
116

FIG. 17


SHKB_CONV
107
SET POINTER TO FIRST CHARACTER
120
"h" OR2 "H"
121
MAP TO "i"
122
"j" OR2 "J"
123
MAP TO "j"
124
"k" OR2 "K"
125
MAP TO "k"
126
"l" OR2 "L"
127
MAP TO "l"
128
"y" OR2 "Y"
129
MAP TO "m"
130
"u" OR2 "U"
131
MAP TO "d"
132
";" OR2 "I"
133
MAP TO ";"
134
"o" OR2 "O"
135
MAP TO "o"
136
"p" OR2 "P"
137
MAP TO "n"
138
"n" OR2 "N"
139
MAP TO "n"
140
"m" OR2 "M"
141
MAP TO "m"
142
"a" OR2 "A"
143
MAP TO "o"
144
"s" OR2 "S"
145
MAP TO "o"
146
"d" OR2 "D"
147
MAP TO "d"
148
"f" OR2 "F"
149
MAP TO "f"
150
N
Y
"g" OR2 "G"
151
MAP TO "t"
152
"q" OR2 "Q"
153
MAP TO "b"
154
"w" OR2 "W"
155
MAP TO ";"
156
"e" OR2 "E"
157
MAP TO "e"
158
"r" OR2 "R"
159
MAP TO "r"
160
"t" OR2 "T"
161
MAP TO "t"
162
"z" OR2 "Z"
163
MAP TO "k"
164
"x" OR2 "X"
165
MAP TO "f"
166
"c" OR2 "C"
167
MAP TO "r"
168
"v" OR2 "V"
169
MAP TO "l"
170
"b" OR2 "B"
171
MAP TO "b"
172
SPACE ?
173
MAP TO "b"
174
PERIOD ?
175
MAP TO "b"
176
MAP TO "?"
177
CONCATENATE MAPPED CHARACTER TO BUFFER
178
SET POINTER TO NEXT CHARACTER
179
NULL ?
180
RETURN
181

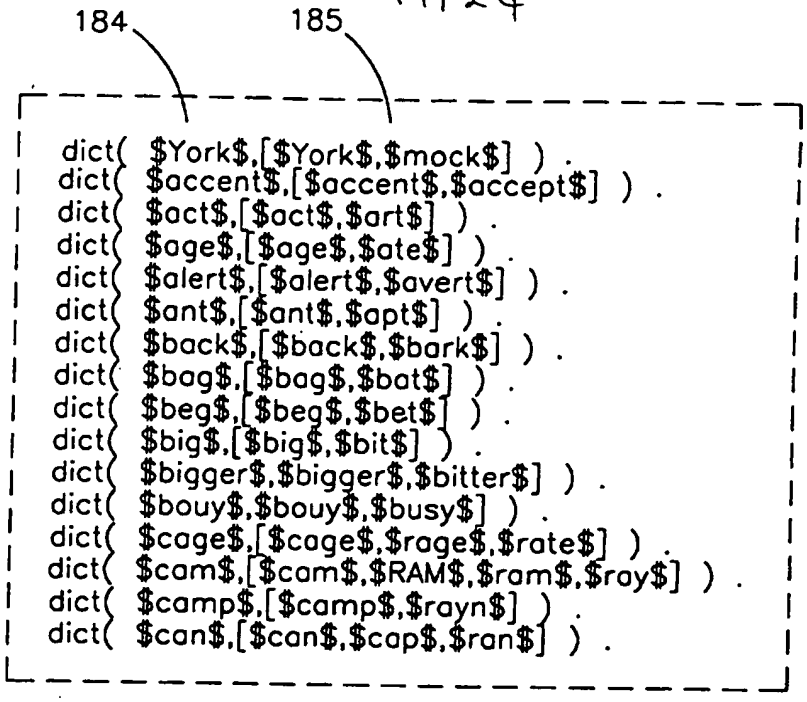


FIG. 18a

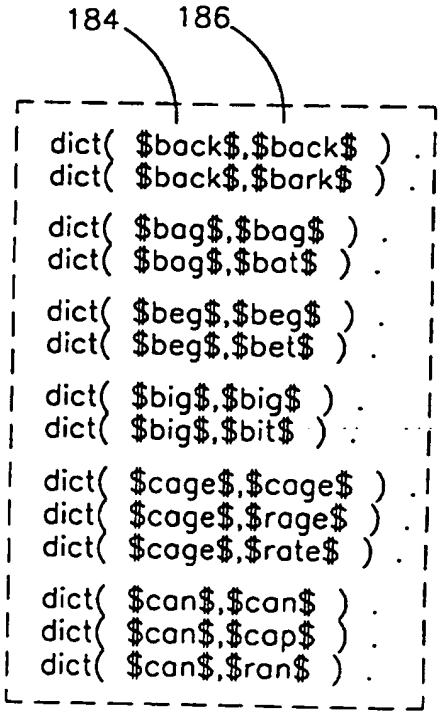


FIG. 18b

FIG. 19

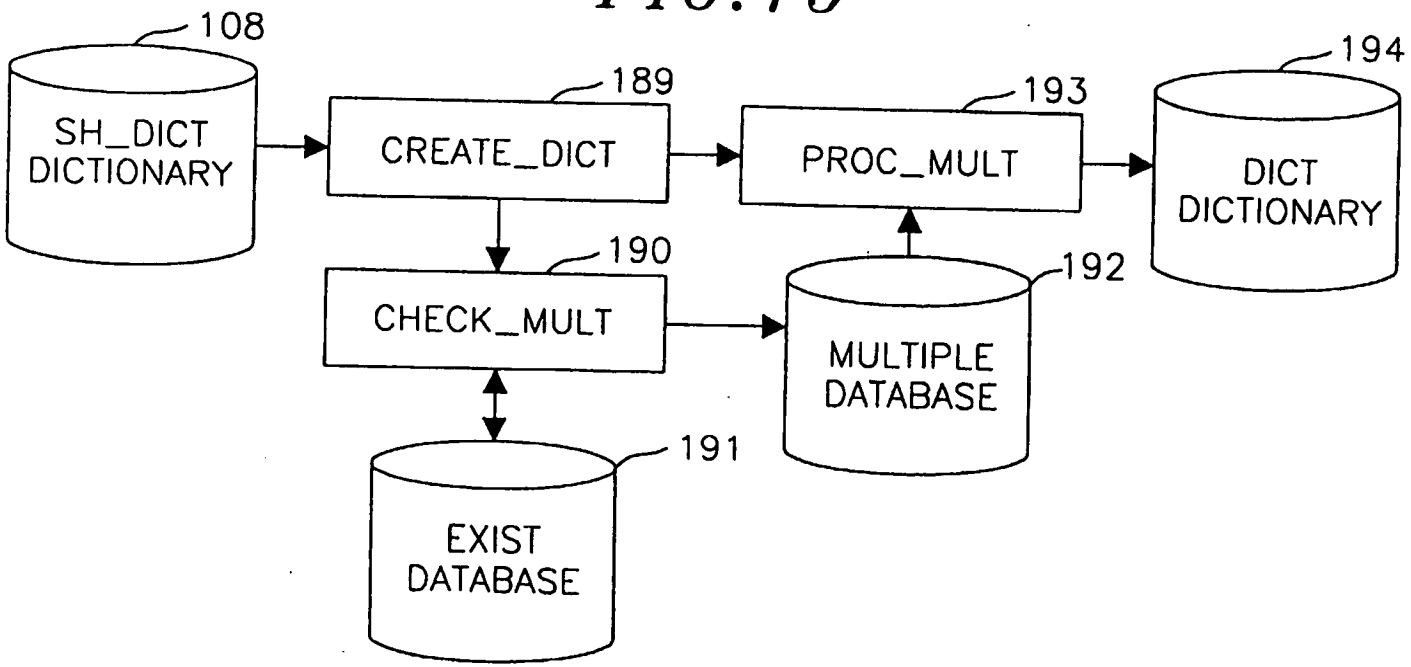

108
SH_DICT
DICTIONARY
189
CREATE_DICT
193
PROC_MULT
194
DICT
DICTIONARY
190
CHECK_MULT
192
MULTIPLE
DATABASE
191
EXIST
DATABASE

FIG. 20

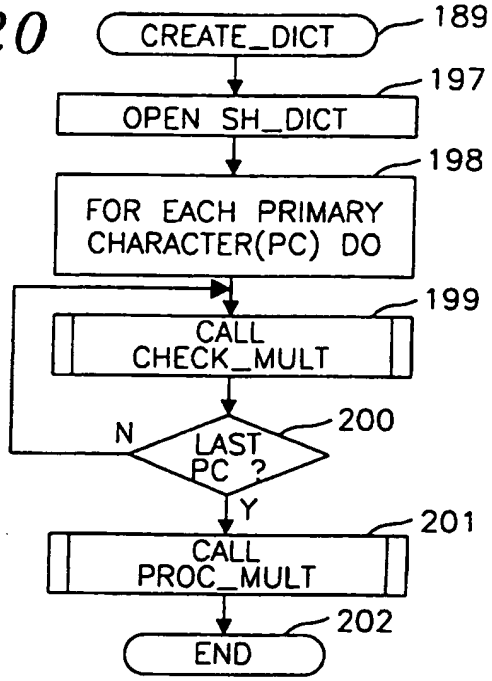

CREATE_DICT
189
OPEN SH_DICT
197
FOR EACH PRIMARY CHARACTER(PC) DO
198
CALL CHECK_MULT
199
LAST PC ?
200
N
Y
CALL PROC_MULT
201
END
202

FIG. 21

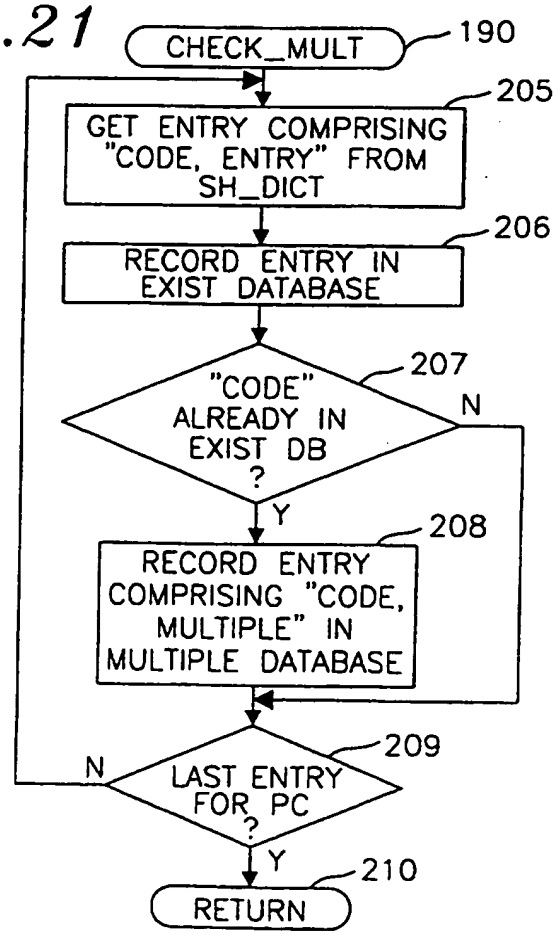

CHECK_MULT
190
GET ENTRY COMPRISING "CODE, ENTRY" FROM SH_DICT
205
RECORD ENTRY IN EXIST DATABASE
206
"CODE" ALREADY IN EXIST DB ?
207
N
Y
RECORD ENTRY COMPRISING "CODE, MULTIPLE" IN MULTIPLE DATABASE
208
LAST ENTRY FOR PC ?
209
N
Y
RETURN
210

FIG.22

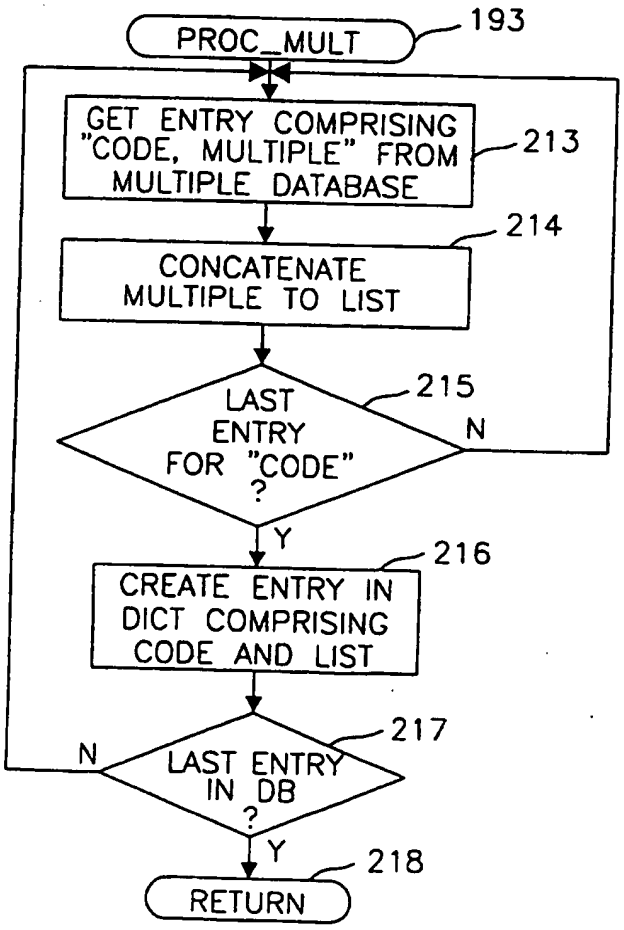

PROC_MULT
193
GET ENTRY COMPRISING "CODE, MULTIPLE" FROM MULTIPLE DATABASE
213
CONCATENATE MULTIPLE TO LIST
214
LAST ENTRY FOR "CODE" ?
215
N
Y
CREATE ENTRY IN DICT COMPRISING CODE AND LIST
216
LAST ENTRY IN DB ?
217
N
Y
RETURN
218

FIG.23a

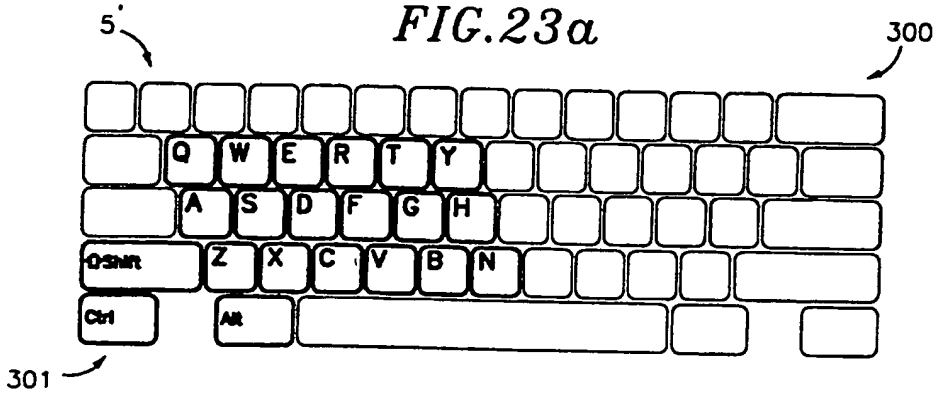
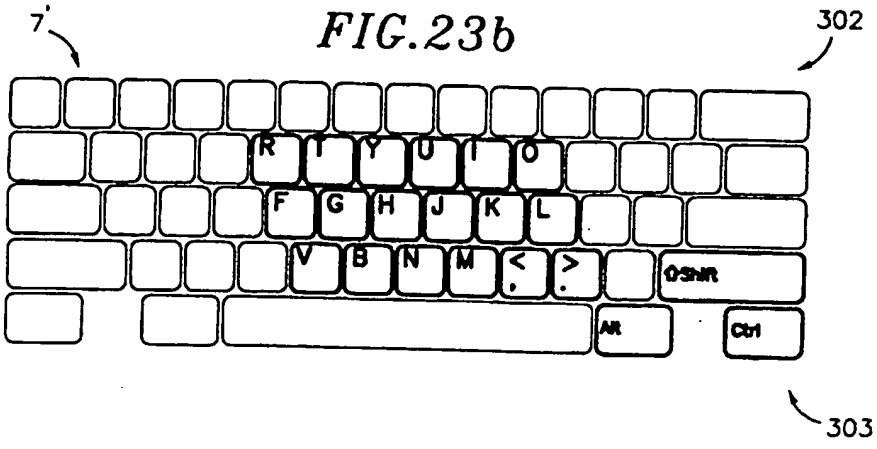


FIG.23b



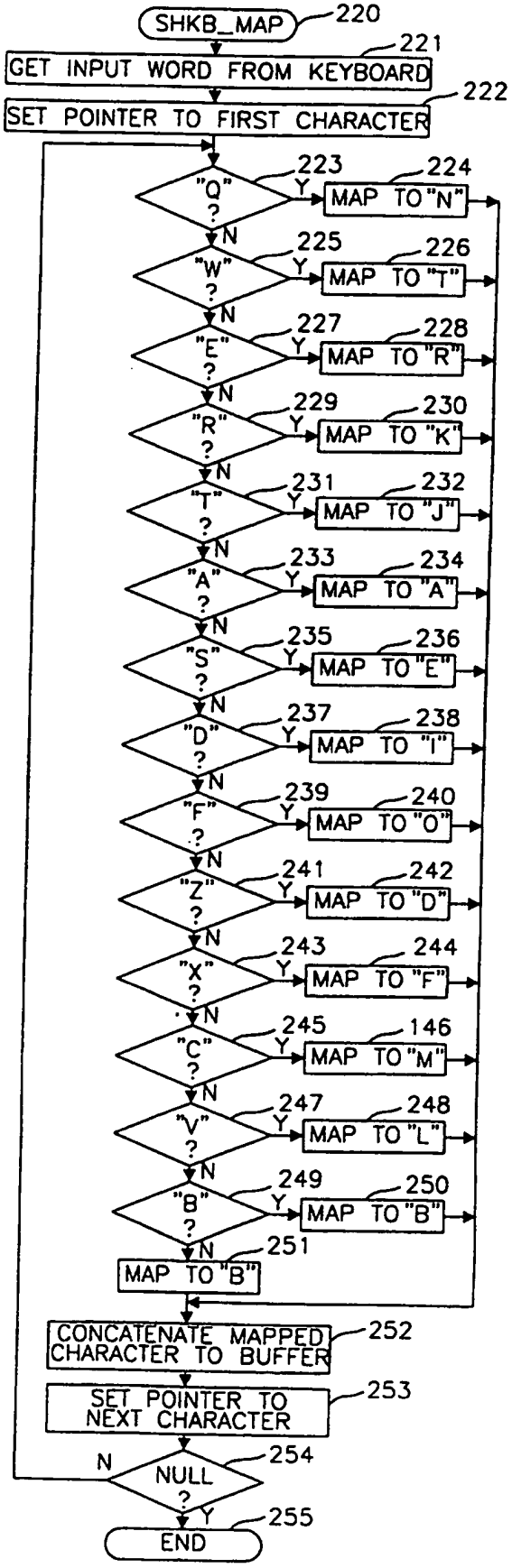

SHKB_MAP
220
GET INPUT WORD FROM KEYBOARD
221
SET POINTER TO FIRST CHARACTER
222
"Q" ?
223
Y
MAP TO "N"
224
N
"W" ?
225
Y
MAP TO "T"
226
N
"E" ?
227
Y
MAP TO "R"
228
N
"R" ?
229
Y
MAP TO "K"
230
N
"T" ?
231
Y
MAP TO "J"
232
N
"A" ?
233
Y
MAP TO "A"
234
N
"S" ?
235
Y
MAP TO "E"
236
N
"D" ?
237
Y
MAP TO "I"
238
N
"F" ?
239
Y
MAP TO "O"
240
N
"Z" ?
241
Y
MAP TO "D"
242
N
"X" ?
243
Y
MAP TO "F"
244
N
"C" ?
245
Y
MAP TO "M"
146
N
"V" ?
247
Y
MAP TO "L"
248
N
"B" ?
249
Y
MAP TO "B"
250
N
MAP TO "B"
251
CONCATENATE MAPPED CHARACTER TO BUFFER
252
SET POINTER TO NEXT CHARACTER
253
N
NULL ?
254
Y
END
255

FIG.24

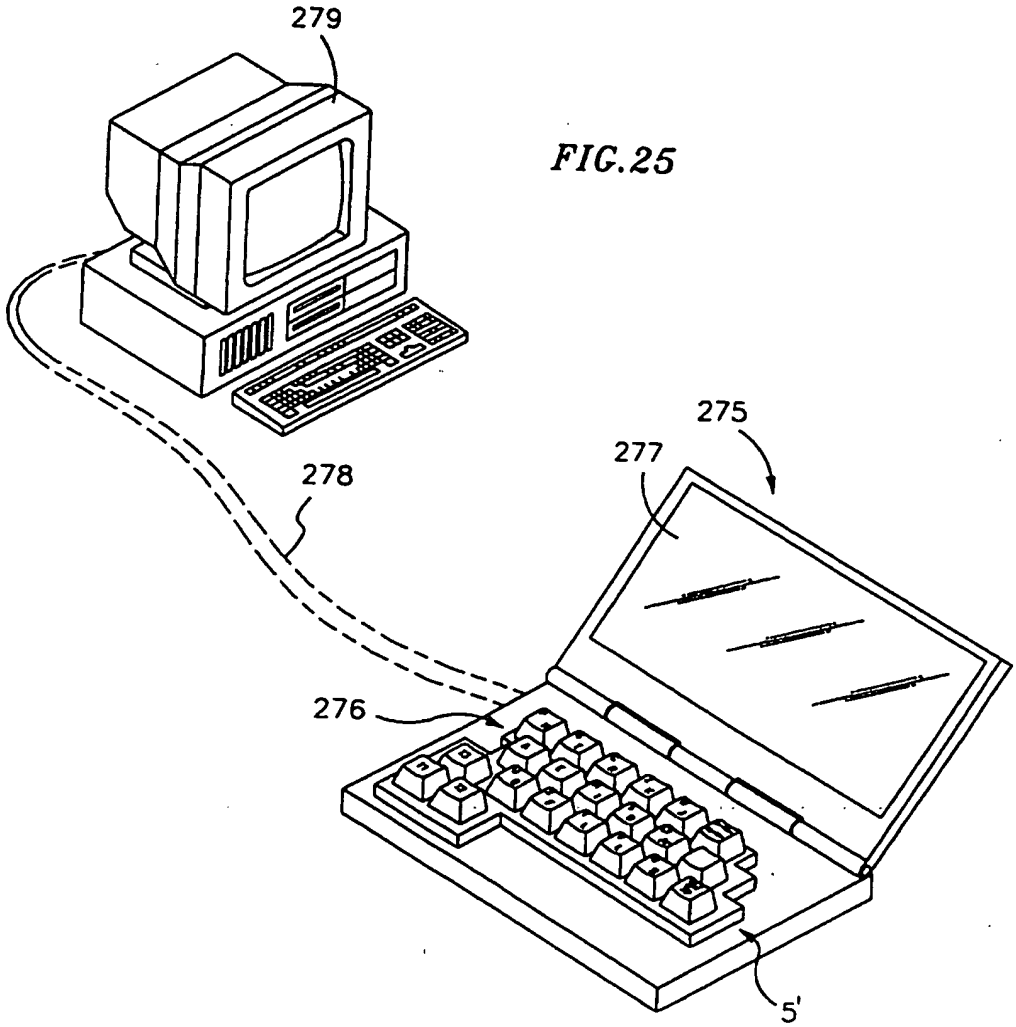
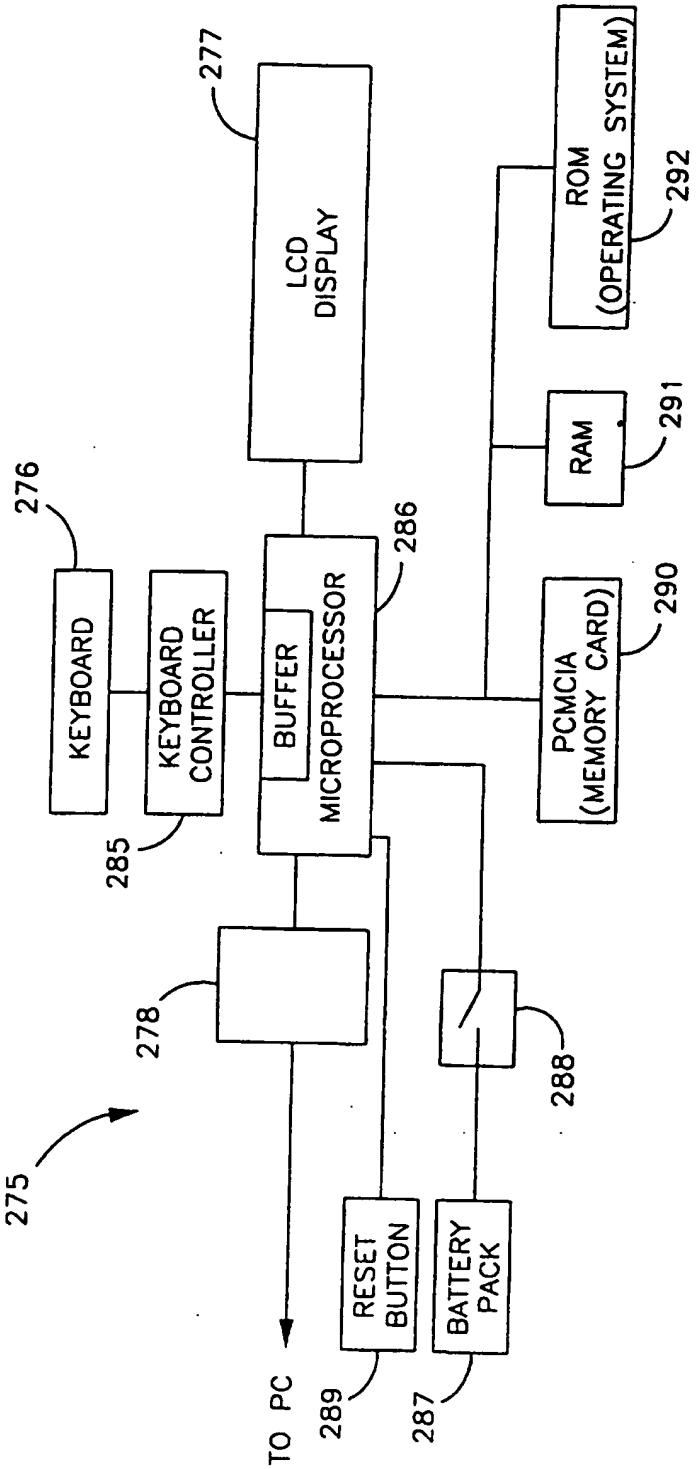


FIG.25

FIG.26


277
LCD DISPLAY
ROM (OPERATING SYSTEM)
292
RAM
291
276
KEYBOARD
KEYBOARD CONTROLLER
BUFFER
MICROPROCESSOR
286
PCMCIA (MEMORY CARD)
290
285
278
288
275
RESET BUTTON
BATTERY PACK
TO PC
289
287

DATA ENTRY SYSTEM

The invention relates to a data entry system comprising a keyboard having multicharacter keys.

Laptop, notebook and palm-top computers have become increasingly popular, in large part because they are extremely portable and can be easily carried to the office, home and abroad. At the same time, the keyboards for these computers have become smaller in size. For example, a typical full-sized keyboard, such as Fujitsu's Model FKD4870 keyboard, is approximately 45 centimetres wide by 17.5 centimetres deep (18 inches by 7 inches). By comparison, the average palm-top computer, such as a Fujitsu's Poget PC Plus Prolinear PS1000, has total outer dimensions of under 25 centimetres wide by 10 centimetres deep (ten inches by four inches), including keyboard, chassis and display assembly. The limiting factor in downsizing keyboards for computers of this kind is the size of the average human hand.

As computers continue to decrease in size, the application of normal touch-typing skills to their cramped dimensions becomes increasingly awkward.

One could create a keyboard incorporating all the functionality of a full-sized keyboard that can be efficiently used with a single hand. A consequence of such an approach would be that keys are assigned more than one character, number, or function to cover the many characters, numbers and functions that the user requires.

Full keyboards are designed with 26 alphabetic character keys corresponding to each character in the Roman alphabet. After the addition of numeric, function and other ancillary keys, the number of keys substantially increases. For instance, the Fujitsu Model FKB4870 keyboard has 101 keys.

Reducing the number of keys to fewer than 20 keys makes it possible to create a keyboard usable by a single hand. However, designing a keyboard with fewer than 26 alphabetic character keys also involves assigning two or more alphabetic characters per key. It may then be desirable to provide means for resolving ambiguities as to which of the two or more alphabetic characters is being selected (i.e. of lifting the inbuilt key degeneracy) whenever such a multicharacter key is used.

A thumb key can be provided to allow the user to select among several alphabetic characters assigned to a multicharacter key. US-A-4 360 892 to Endfield discloses a portable word processor having a single-hand keyboard with finger and thumb keys. These keys are entered in chords to enter a character, number, or punctuation mark but the thumb keys are not used to select a particular character once the operator has chosen a keyboard by using one of the thumb keys.

US-A-4 042 777 to Bequaert et al. discloses a single-hand keyboard with finger and thumb keys. The operator presses thumb keys to select an alphabet or case and can press up to four finger keys with one finger to uniquely identify a character within the alphabet or case. The operator does not press thumb keys to select a particular character once he has chosen the alphabet or case.

US-A-5 288 158 to Mathias discloses a single-hand keyboard that comprises keys for representing one-half of a full keyboard. The finger keys are used to key in two characters, one from each half of the full keyboard. The operator presses a modifier key with his thumb to choose between the two halves of the keyboard. With the approaches taken in US-A-4 360 892, US-A-4 042 777 and US-A-5 288 158, the user must decide whether to use the thumb key for most keystrokes.

To distinguish between two or more alphabetic characters entered using a multicharacter key, one could resolve ambiguities post hoc. US-A-4 484 305 to Ho discloses a phonetic word processor that enables a user to enter Chinese, Japanese, or other ideographic characters using a standard QWERTY keyboard. The user selects the three word components, namely consonants, vowels and tones. An ideographic character along with any characters having the same sound (homonyms) is displayed on the screen.

US-A-4 684 926 to Yong-Min discloses a system of encoding Chinese characters that includes a keyboard comprising an arrangement of basic strokes constituting roots of Chinese characters and phrases encoded according to their geometric forms. The user toggles between characters having the same encoding. The approach taken in US-A-4 484 305 and US-A-4 684 926 allows the resolution of ambiguity between individual ideographic characters or homonyms but not between multiple-character words.

According to a first aspect of the invention there is provided a data entry system comprising: keys each assigned a character, the keys comprising multiple character keys, each multiple character key being assigned multiple characters;

a character generator for generating upon entry of a sequence of the keys, a sequence of the characters comprising a character assigned to each of the sequence of the keys, there being an ambiguity among the multiple characters assigned to a multiple character key that is entered as to the correct assigned character that should be included in the sequence of characters; and

a multiple character resolver responsive to the sequence of keys that are entered for resolving the ambiguity.

According to a second aspect of the invention, there is provided a data entry system for resolving ambiguity in a word input thereto comprising a keyboard having a plurality of keys suitable for keying in words, each of these keys having a primary character and at least one of these keys having a plurality of alternate characters, an input word being such that it may be ambiguously spelt;

logic for accepting an input word from the keyboard;

logic for matching an input word to a list comprising one or more correctly spelt words; and

a display for displaying one or more correctly spelled words to a user in the event that an input word is ambiguously spelt.

According to a third aspect of the invention, there is provided a data entry system according to the first or second aspect of the invention and comprising:

a plurality of keys, at least some of the keys being assigned at least one character and at least one key being assigned a plurality of characters;

a character buffer for storing a plurality of character keycodes;

logic for obtaining a character keycode from the plurality of keys;

logic for returning a word corresponding to the contents of the character buffer when the character keycode corresponds to a carriage return key;

logic for truncating the contents of the character buffer by one character when the character keycode corresponds to a backspace key;

logic for setting the contents of the character buffer to a null string when the character keycode corresponds to an escape key;

logic for performing ambiguity resolution on the contents of the character buffer when the character

keycode corresponds to an ambiguity resolution key; and

logic for storing the character keycode in the character buffer when the character keycode does not correspond to a carriage return key or a backspace key or in escape key or an ambiguity resolution key.

Preferably the ambiguity resolution is performed with word variation logic.

Further aspects and embodiments of the invention are exemplified by the attached claims as well as by the following:

A further aspect of the invention is a keyboard for operation by fingers of a single hand of a person. Character keys, each of which when operated enables generation of an assigned character. A first bank of the keys has at least one key assigned the characters *n* and *p* and at least one key assigned the characters *t* and *g* disposed for operation by the index finger, at least one key assigned the characters *r* and *c* disposed for operation by the middle finger, at least one key assigned the characters *k* and *z* disposed for operation by the ring finger and at least one key assigned the characters *j* and *w* disposed for operation by the little finger of the single hand. A second bank of the keys has at least one key assigned the characters *d* and *u* and at least one key assigned the characters *f* and *x* disposed for operation by the index finger, at least one key assigned the characters *m* and *y* disposed for operation by the middle finger, at least one key assigned the characters *l* and *v* disposed for operation by the ring finger and at least one key assigned the characters *b* and *q* disposed for operation by the little finger of the single hand. A middle bank of the keys between the first and second banks has at least one key assigned the character *a* and at least one key assigned the character *e* disposed for operation by the index finger, at least one key assigned the characters *i* and

h disposed for operation by the middle finger and at least one key assigned the characters *o* and *s* disposed for operation by the ring finger of the single hand.

Preferably, the keyboard has keyboard function keys disposed for operation by a finger of the single hand and the middle bank further has at least one keyboard function key operable in the keyboard for control of capitalization of the character being generated and disposed for operation by the little finger.

Preferably, the home bank has at least one keyboard function key disposed for operation by the little finger for causing resolution of the ambiguity between the generated characters when, during operation of a plurality of the keys, at least one of the character keys is operated that is assigned more than one of the characters.

Preferably, the second bank has at least one keyboard function key for entry of a space character between the characters being generated and disposed for operation by the little finger.

Additionally, there is preferably provided a secondary bank of keys disposed for operation by a thumb of the single hand for control of the keyboard.

Preferably, the keys are arranged in substantially parallel rows and diagonal columns. The diagonal columns have a first diagonal column with the keys assigned the characters *n*, *p*, *a*, *d* and *u* a second diagonal column with the keys assigned the characters *t*, *g*, *e*, *f* and *x* a third diagonal column with the keys assigned the characters *r*, *c*, *i*, *h*, *m* and *y* a fourth diagonal column with the keys assigned the characters *k*, *z*, *o*, *s*, *l* and *v* and a fifth diagonal column with the keys assigned the characters *j*, *w*, *b* and *q*.

Preferably, there is a sixth diagonal row with at least one function key disposed for operation by the

little finger for causing resolution of the ambiguity between the characters to be so generated when one of the character keys is operated, during operation of a plurality of the keys, that is assigned more than one of the characters.

Preferably, the keyboard has character keys laid out and disposed substantially as shown in FIG. 2 or FIG. 3.

Another aspect of the invention is a keyboard for entering, with the fingers of a single hand, the letters of the alphabet. The keyboard has first, second and middle rows of letter entry keys, the middle row being located between the first and second rows. In the middle row, the letter entry keys are, in sequential order, "a" key, "e" key, "i" and "h" key and "o" and "s" key. In the first row, the keys are, in sequential order, "n" and "p" key, "g" and "t" key, "c" and "r" key, "z" and "k" key and "w" and "j" key. In the third row, the keys are, in sequential order, "d" and "u" key, "f" and "x" key, "m" and "y" key, "l" and "v" key and "b" and "q" key.

Preferably, there is a representation of each of the recited letters displayed in association with the corresponding key and in a preferred embodiment, each such representation of each letter is on the corresponding key.

In a preferred embodiment, the keys are arranged in substantially parallel rows and diagonal columns. The diagonal columns have a first diagonal column with the *n, p, a, d* and *u* keys a second diagonal column with the *t, g, e, f* and *x* keys a third diagonal column with the *r, c, i, h, m* and *y* keys a fourth diagonal column with the *k, z, o, s, l* and *v* keys and a fifth diagonal column with the *j, w, b* and *q* keys.

Preferably, the fifth diagonal column and the middle row have a function key for control of the

keyboard and a representation "Ar" displayed in association with such control key.

Yet another aspect of the invention is a character generator for a keyboard where the keyboard has keys for entry of the characters. Each key is assigned a character and each of at least some of said keys are multiple-character keys assigned multiple characters.

The generator generates one of the characters assigned to each of a sequence of the character keys that are entered on the keyboard, there being an ambiguity as to the correct character to be generated when any said multiple-character key is entered in the sequence of keys. An ambiguity resolver operates based on the sequence of character keys that have been entered for resolving the correct character for any of the multiple-character keys in the sequence of keys that are entered.

A still further aspect of the invention is a keyboard character entry system having keys each assigned a character, the keys include multiple-character keys each assigned multiple characters. A character generator generates, upon entry of a sequence of the keys, a sequence of the characters including a character assigned to each of the sequence of the keys, there being an ambiguity among the multiple characters assigned to a multiple-character key that is entered as to the correct assigned character that should be included in the sequence of characters. A multiple-character resolver is responsive to the sequence of keys that are entered for resolving the ambiguity.

A yet further aspect of the invention is a method for generating character sequences using a character generator for a keyboard where the keyboard has keys for entry of assigned characters. Each of at least some of said keys is a multiple-character key assigned

multiple characters. The generator is used for generating one of the characters assigned to each of a sequence of the character keys that are entered on the keyboard. There is an ambiguity as to the correct character to be generated when any multiple-character key is entered in the sequence of keys. Based on the sequence of character keys that have been entered, a resolution is made as to the correct character for any of the multiple-character keys in the sequence of keys that are entered.

Accordingly, it will be appreciated that a keyboard embodying the invention can be provided which is able to resolve ambiguities between multiple-characters assigned to the same key when a sequence of keys is entered. Moreover, in a preferred embodiment a single-handed keyboard can be provided which is able to resolve ambiguities between multiple-characters assigned to the same key when a sequence of keys is entered, there being more than one character assigned per key. Furthermore, in another preferred embodiment, there is provided a keyboard which has keys laid out and placed in the keyboard to maximise efficiency of entry by the fingers of a single hand.

For a better understanding of the invention, and to show how the same may be carried into effect, reference will now be made, by way of example, to the accompanying drawings, in which:-

FIG. 1 depicts an isometric view of a single-hand keyboard, including a character generator, with no thumb keys and embodies the present invention;

FIG. 2 is an isometric view of a single-hand keyboard, including a character generator, having a key layout for a right-handed single-hand keyboard with thumb keys and embodies the present invention;

FIG. 3 is an isometric view of a single-hand

keyboard, including a character generator, having a key layout for a left-handed single-hand keyboard with thumb keys and embodies the present invention;

FIG. 4 is an isometric view of the keyboard of FIG. 2 showing the key layout for a control character set;

FIG. 5 is an isometric view of the keyboard of FIG. 2 showing the key layout for a numeric character set;

FIG. 6 is an isometric view of the keyboard of FIG. 2 showing the key layout for a function key character set;

FIG. 7 is an isometric view of the keyboard of FIG. 2 showing the key layout for a terminal character set;

FIG. 8 is a block diagram of a computer program forming ambiguity resolution logic for use in a programmed computer in the keyboards of FIGS. 1 and 2 or the system of FIG. 26;

FIG. 9 is a flow diagram of the MAIN routine of FIG. 8;

FIG. 10 is a flow diagram of the GET_CODE routine of FIG. 8;

FIG. 11 is a flow diagram of the GEN_CAP routine of FIG. 8;

FIG. 12 is a flow diagram of the AMB_RES routine of FIG. 8;

FIG. 13 is a flow diagram of the ENH_AMB_RES routine of FIG. 8;

FIG. 14 is a diagram depicting the structure of the SH_DICT dictionary of FIG. 8;

FIG. 15 is a block diagram showing the process for creating the SH_DICT dictionary of FIG. 8;

FIG. 16 is a flow diagram of the CREATE_SH_DICT routine of FIG. 8;

FIG. 17 is a flow diagram of the SHKB_CONV routine

of FIG. 8;

FIG. 18, including FIGS. 18a and 18b, depict diagrams showing the structure of the DICT dictionary, a preferred embodiment being shown in FIG. 18a and an alternate embodiment being shown in FIG. 18b;

FIG. 19 is a block diagram depicting a process for creating the DICT dictionary;

FIG. 20 is a flow diagram of the CREATE_DICT routine of FIG. 19;

FIG. 21 is a flow diagram of the CHECK_MULT routine of FIG. 19;

FIG. 22 is a flow diagram of the PROC_MULT routine of FIG. 19;

FIG. 23, including FIGS. 23a and 23b, are plan views depicting the key layout of a conventional two-handed keyboard converted for single-hand use by a programmed computer. A right-handed keyboard is depicted in FIG. 23a and a left-handed keyboard is depicted in FIG. 23b;

FIG. 24 is a flow diagram of SHKB_MAP routine of a computer program for use in the programmed computer referred to in the description of FIG. 23 for emulating a single-handed keyboard using the conventional two handed keyboard;

FIG. 25 is a perspective view of the dedicated single-hand keyboard corrected as an input device to a personal computer; and

FIG. 26 is a schematic block diagram depicting the electronic components for a single-hand keyboard systems, including a programmed computer or processor.

An embodiment of the present invention is a keyboard system such as depicted in FIG. 26. A keyboard 276 with keys and key switches (not shown) sends binary coded digital signals representative of each key that is struck or entered to a keyboard controller 285. Keyboard controller 285, in turn,

ELEC
#2091
STAIR #3
ST-1
UP
DN
OPEN TO STRUCTURE
CAFE

sends signals unique to the keys which have been entered to a digital computer or microprocessor 286. The microprocessor 286, under control of a computer program, recognizes the entered keys and forms or generates a character or symbol assigned to the keys that have been entered in the form of digital coded signals. A visual display of the character or symbol is also formed on a visual display 277.

Although an embodiment of the invention is useful for conventional two-handed keyboards, it is especially useful for a single-hand keyboard. The discussion to follow is presented in three sections covering the spatial arrangement of the keys, the ambiguity resolution logic and the software emulator and hardware embodiments.

I. Spatial Arrangement of keys

Referring to FIGS. 1, 2 and 3, three different single-hand keyboards are illustrated. FIG. 1 shows the general layout of a right-handed single-hand keyboard 1 comprising a keypad 1 having an upper row 2, a home row 3 and a lower row 4 of finger keys, forming first, middle and second rows of keys. By way of example, there are six finger keys per row for a total of 18 finger keys. The keys in rows 2, 3 and 4 are character keys arranged in columns a, b, c, d, e and f. Columns a through e have keys each assigned for generating one or more characters. An exception is the key in row 3 of column f, which is a function key. The keys in column f of each row are function keys for controlling keyboard functions. A bank of three thumb keys are included in the embodiments of FIGS. 2 and 3 but not of FIG. 1. The embodiments of FIGS. 2 and 3 also have a bank of thumb keys 6.

The keyboard can be configured with either right-hand keypad 1' as in FIG. 2 or left-hand keypad 1" as in FIG. 3 and the specific key layout for one is

preferably a mirror image of the other. Referring to FIG. 2, right-handed single-hand keyboard 5 is shown with three right hand thumb keys 6.

A. Assignment of Keys

FIGS. 2 and 3 are labelled to show characters and the control and functions assigned to each key for the right-handed single-hand keyboard 5. The functionality, layout of components and operation of the right-handed single-hand keyboard 5 and the left-handed single-hand keyboard 7 are essentially the same and therefore the description of one will be understood to apply equally to the other. The thumb keys 6 function in a manner similar to conventional shift keys. Each character key has a primary character assigned to it. Some finger keys have a second alternate character or special function assigned to it. Thumb keys are depressed simultaneously with a finger key to access an alternate character or to execute a special function.

In FIG. 2, the 26 characters of the Roman alphabet are shown assigned to 14 finger keys in columns a-f of rows 2, 3 and 4 with only two of those characters being assigned to a single key. Those are the keys for the characters A and E in columns a and b of row 3. The remaining 12 finger keys are multicharacter keys with two characters assigned to each.

The lower character assigned to and shown on each multi-character key is the primary character which is formed when the user presses that multicharacter key. The upper character of each multicharacter key is an alternate character which is registered when the user simultaneously enters the "T2" thumb key and that multicharacter key. Thus, the set of primary characters comprises the characters A, B, D, E, F, I, J, K, L, M, N, O, R and T. The set of alternate characters comprises the characters C, G, H, P, Q, S,

U, V, W, X, Y and Z. In the following discussion, the finger keys will be referred to by their primary character assignment unless otherwise indicated.

The alphabetic key assignments proceeding from left to right for the right-handed keyboard/right to left for the left-handed keyboard in order of primary/alternate characters, are for the upper row 2: N/P, T/G, R/C, K/Z and J/W; for the home row 3; A, E, I/H and O/S; and for the lower row 4; D/U, F/X, M/Y, L/V and B/Q. Ambiguity resolution logic, to be described, resolves the ambiguity caused by the dual use of each multicharacter key for more than one character.

There are four non-alphabetic finger control keys. In the upper row 2, the right-most key in column f of row 2 is the "Up-Ctrl/En-Num key. The primary function assigned is the capitalization control function, "Up-Ctrl: Lower case characters are normally generated The "Up-Ctrl" key causes upper case or capitalized characters to be generated. The alternate function En-Num causes the keyboard to enter or exit a numeric character entry mode, En-Num, in which strokes of the keys entered cause numerals to be formed instead of alphabetic characters. To access the alternate function, the user simultaneously depresses the "T2" thumb key and the "Up-Ctrl/En-Num" key.

The second key from the right in the home row 3 is the "Ar/Cr" control key and is the only key in column e that is not a character key. The primary function Ar triggers the ambiguity resolution logic. The alternate function Cr enters a carriage return. To have access to the alternate function, the user simultaneously depresses the "T2" thumb key and the "Ar/Cr" key.

The right-most key, column f of the home row 3, is the "Tc/Ctrl" key. The primary function Tc, causes formation of a terminal character which is a

punctuation mark. The set of available terminal characters is described below. The alternate function Ctrl enables a control function, to be entered. To access the alternate function, the user simultaneously depresses the "T2" thumb key and the "Tc/Ctrl" finger key.

The right-most key in column f, column f of the lower row 4 is the "Sp/Shf" key. The primary function Sp enters a space. The alternate function Shf enables the user to input special codes such as case shifts and cursor-up movements. To access the alternate function, the user simultaneously depresses the "T2" thumb key and the "Sp/Shf" key.

For ease of use of the single-hand keyboard, the home row key assignments for the right hand are: index finger on the "E" key, middle finger on the "I" key, ring finger on the "O" key, little finger on the "Ar" key and thumb on the "T2" key.

B. Basis for Alphabetic Character Key Assignments

The assignment of the alphabetic characters to the finger keys comprising the single-handed keyboard is a function of three variables.

The first variable is the frequency of occurrence for each alphabetic character in ordinary English text. The frequency of character occurrences is shown in Table 1 and is detailed in W.A. Beeching, The Century of the Typewriter, pp. 41-42 (London 1974).

Frequency of Appearance for Alphabetic Characters in ordinary English Text

E	T	A	O	N	R	I
S	H	D	L	F	C	M
U	G	Y	P	W	B	V
K	X	J	Q	Z		

Table 1

Referring to Table 1, the 26 characters of the Roman alphabet are shown in decreasing order of frequency from left to right, top to bottom. Thus, the seven most frequently appearing characters are E, T, A, O, N, R and I on keys in the middle and upper left of the keyboard of FIG. 2.

The second variable in the assignment of keys in the single-hand keyboard is the relative ease of movement of the fingers. For each finger, ease of movement is defined in terms of three variables: lateral movement, flexion and extension. Lateral movement refers to the relative ability of a finger to move in a horizontal, left-to-right manner. Accordingly, the index and little fingers enjoy the most lateral movement since both are "outer" fingers. Conversely, the middle and ring fingers have the least lateral movement because both are constrained on either side by other fingers: the index and ring fingers surround the middle finger and the middle and little fingers surround the ring finger. Consequently, the single-hand keyboard assigns two columns of keys apiece to the ring finger and little finger and one column of keys apiece to the middle and ring fingers.

Flexion refers to the ability of a finger to curl. Normally, all fingers of the hand are equally flexible. Extension refers to the ability of a finger to stretch outwardly. Normally, all fingers are also equally extendable. However, in key layout design, the effects of flexion and extension are affected by finger length. Thus, although the little finger enjoys a greater degree of lateral movement than either the middle or ring finger, it is the least suitable finger for movement away from the home row due to its short length. Moving the little finger away from the home row requires movement of the entire hand, an

undesirable side effect. Nevertheless, the little finger is well equipped for timing the movements of fingers and for helping to anchor the other fingers to the home row position. As a result, in the single-hand keyboard, the home row position for the little finger is the "Ar" key which triggers the ambiguity resolution logic, a frequently used function in the preferred embodiment.

The keyboard of FIG. 2 assigns alphabetic characters in accordance with the relative ease of movement in fingers and the frequency of occurrence of each alphabetic character in ordinary text. Since the index finger enjoys the greatest overall ease of movement, four of the most frequently used characters, A, E, N and T, are assigned to that finger. In addition, the characters D and F are also assigned to the index finger, though in the lower row positions.

The most frequently used alphabetic character, E, is assigned to the home row position for the index finger. The second most frequently used alphabetic character, T, is assigned to the upper row position of the index finger. The third most frequently used alphabetic character, A, is assigned to the left position of the index finger. Finally, the fifth most frequently used alphabetic character, N, is assigned to the upper left position of the index finger. The fourth most frequently used character, O, is assigned to the home row position for the ring finger. The sixth and seventh most frequently used characters, R and I, are assigned to the upper row and home row positions for the middle finger.

The most frequent run of alphabetic characters is "TION." The key layout places the T, I and O keys in positions conducive to a more natural flow of finger movement, proceeding from left to right for the right hand and from right to left for the left hand.

Moreover, it is easier to type this sequence of characters if the character I is assigned to a home row position. Thus, although the character R appears more frequently than the character I, the character I appears in the home row position for the middle finger for typing efficiency.

Note that in the preferred embodiment, three of the five vowels in the English language, E, I and O, are assigned to home row positions as primary characters and do not require a thumb key for entry. Furthermore, the most frequently used consonants, N, R and T, are assigned as primary characters and also do not require a thumb key for entry.

The third variable in the assignment of keys in the single-hand keyboard is the lessening of the multiplicity of words forming a raw input character string. Consequently, the key arrangement seeks to minimize the number of multiple word choices that can be entered by using the primary characters only, thereby enhancing the effectiveness of the ambiguity resolution logic. Moreover, the remaining alphabetic characters are assigned by the order of their frequency of appearance in usual text.

C. Character Sets

Referring to FIG. 4, the same set of finger keys are displayed in keyboard 5 as in FIG. 2, but some are labelled with their control functions for ease of understanding. The control keys are typically for cursor movement in a display and control functions, but other purposes would be readily apparent to one skilled in the art. A control character is entered by simultaneously entering the "T1" thumb key and one of the control finger keys.

Referring to FIG. 5, the same set of finger keys are displayed in keyboard 5 as in FIG. 2, but each is labelled with a numeric character or mathematical

operator function. These characters are used for entering numbers and mathematical operations wherein the functionality is provided through computer programs in a processor (such as microprocessor 285 of FIG. 26) that recognizes the significance of the respective numeric character or mathematical operator using methodologies known to one skilled in the art. A character from the numeric character set can be entered in two ways. The single-hand keyboard supports a numeric character entry mode. This mode is toggled by simultaneously entering the "T2" thumb key and the "Up-Ctrl/En-Num key. Once toggled, the keys labelled with numbers in FIG. 5 become number keys and a numeric character is entered by entering one of the numeric keys labelled. Second, a numeric character is also entered by simultaneously entering the "T1" thumb key and a finger key with the desired number as shown in FIG. 5. This manner of entering a numeric character operates independently of the numeric character entry mode.

Referring to FIG. 6, the same set of keys are displayed in keyboard 5 as in FIG. 2, but some are labelled F1 through F12 for their function key functions. Function keys can be assigned by a user or programmer for various purposes as typically done with normal keyboards by changing the computer program function assigned to the keys. A function key from this set can be entered by simultaneously entering the "T3" thumb key and the "Tc/Ctrl" key, followed by one of the function keys F1 - F12.

Referring to FIG. 7, the same set of keys are displayed as Keyboard 5 as in FIG. 2, but some are labelled with the symbols ' " : % ; ^ , \ $ @ ([ )] & 1 ! ~ < { > } ? = - for purposes of this application with their terminal character functions. The set of terminal characters consists of punctuation marks and

each terminal character is entered by entering the "Tc/Ctrl" key, followed by the corresponding key.

II. Ambiguity Resolution Logic

The keys of the keyboard are preferably labelled, as in FIGS. 2 and 3. The ambiguity resolution logic allows the user to select the desired character assigned to a multi-character key using the "T2" thumb key as described above. Alternatively, the ambiguity resolution logic can be used to select and display a sequence of characters that make up an actual word after the sequence of character keys bearing the desired characters are entered. In the latter case, if the desired sequence of characters or word is not generated and displayed to the user, the user can select one or more alternate character sequences or words by striking character sequence or assigned to the same sequence of entered keys by entering the "Ar" key until the desired sequence of characters or word is generated. Each time the "Ar" key is entered, another sequence of characters or word is selected and displayed using the characters assigned the keys that were previously entered.

To explain further, the character to be used caused by the assignment of multiple alphabetic characters to one key is an "ambiguity." The underlying premise is straightforward: by allowing a user to enter a character string without having to simultaneously enter a thumb key to select the alternate alphabetic character that is assigned to that multicharacter key, the user can maintain a faster and more efficient typing pace than if otherwise interrupted by the inconvenience of entering a thumb key. Ambiguity resolution logic enables a user to enter a sequence of keys and hence generate a sequence of characters for viewing on a display and to thereby defer resolving the character ambiguities inherent therein by virtue of

exclusive use of primary character keys.

By way of example, assume that the user decides to input the word "this." Referring to FIG. 2, the user enters the word by striking the keys with the characters "T-H-I-S". Direct selection of the characters H and S require the "T2" thumb key to be entered simultaneously with the I/H and O/S keys. Conversely, the user can strike the keys with the characters T-H-I-S, also T-I-I-O, which will cause the characters "tiio" to be generated in digital form and then displayed. Entry of the "Ar" key triggers the ambiguity resolution logic, which causes display of the word "this." The ambiguity resolution logic ascertains that the user really meant to enter a word with its second character being *h* and its fourth character being *s* instead of the characters *i* and *o*, as was actually registered by the keyboard. Thus, the user saves effort by avoiding having to enter the "T2" thumb key twice for this sequence of character keys. This operation will now be discussed in more detail.

A. Basic Ambiguity Resolution Logic

Character string ambiguities are resolved on demand. To invoke the ambiguity resolution after a string of keys have been entered, such as for a complete word, the user depresses the "Ar" key. Entry of the "Ar" key causes the ambiguity resolution logic to present to the user a sequence of characters or a word from a list of word choices corresponding to the possible spellings of words enterable by the user based on the multicharacter keys entered. If the new sequence of characters or word is not the desired one the "Ar" key can be entered time after time until the desired sequence of characters or word is generated and displayed.

Referring to FIG. 8, an overall block diagram of the computer program or ambiguity resolution logic that

is to be run on the computer system of FIG. 26 is illustrated. The system is implemented on a microprocessor system with memory and comprises five functional modules and two data dictionaries. The point of entry is the MAIN routine 20, which calls the GET_CODE routine 21, the GEN_CAP routine 22 and the AMB_RES routine 23. In turn, the AMB_RES routine 23 accesses the SH_DICT dictionary 24 and calls the ENH_AMB_RES routine 25.

The second stage of ambiguity resolution logic is embodied in the ENH_AMB_RES routine 25. Like the AMB_RES routine 23, the ENH_AMB_RES routine 25 accesses the dictionary, the DICT dictionary 26. The ENH_AMB_RES routine 25 also calls the GET_CODE routine 21 and the GEN_CAP routine 22. The GEN_CAP routine 22 also calls the GET_CODE routine 21. The precise functionality of the foregoing routines will now be described in more detail.

Referring to FIG. 9, a flow diagram for the MAIN routine 20 is illustrated. The purpose of the MAIN routine 20 is to return a single input word entered by the user. The routine accepts one input keycode at a time by iteratively calling the GET_CODE routine 21 (block 27). A keycode is a numeric value that uniquely identifies a character for entry of the key that was inputted via the keyboard. Keycodes are required because multiple characters are represented on each key. The ASCII character set comprises a commonly known set of keycodes.

After accepting an input keycode, the MAIN routine 20 parses that keycode to determine the appropriate action to take (blocks 28-35). The routine maintains a buffer (see FIG. 26) in which it assimilates the set of input keycodes received from the GET_CODE routine 21. If the input keycode is a carriage return (block 28), the routine is exited (block 38). If the input keycode

is a backspace (block 29), the buffer of input keycodes is truncated by one keycode which effectively "erases" the last character typed (block 30). The MAIN routine 20 will then accept a new keycode and begin the parsing process anew. If the input keycode is an escape (block 31), the buffer of input keycodes is set to an empty, or null, string (block 32) and the MAIN routine 20 is exited (block 38). If the input keycode is an ambiguity resolution character (block 35), that is, the "Ar" key was entered, the AMB_RES routine 23 is called (block 36) and the MAIN routine 20 is exited (block 38). Finally, if the input keycode is neither a carriage return, backspace, escape, or ambiguity resolution character, the buffer of input keycode is concatenated with the input keycode (block 37), the MAIN routine 20 accepts the next input keycode and the parsing process begins anew.

Referring to FIG. 10, a flow diagram for the GET_CODE routine 21 called at block 27 of the MAIN routine 20 is illustrated. The primary purpose of the GET_CODE routine 21 is to accept one or more raw input characters from the keyboard and to return to the calling routine a single keycode corresponding to a single character from one of the five character sets supported by the single-hand keyboard (alphabetic, control, numeric, function key and terminal character sets).

The GET_CODE routine 21 begins by accepting a raw input character selected by a keystroke on the keyboard (block 40). The MAIN routine 20 maintains an internal indication of whether the keyboard is currently in a numeric character entry mode. This internal indication (not shown) is checked by the GET_CODE routine 21 (block 41) and if the indication is presently set, the routine uses the value of the input character to look up the numeric character keycode (block 42) from an

internal table (not shown) and returns that keycode to the calling routine MAIN (block 67). If the internal indication is not set, the GET_CODE routine 21 parses the raw input character to determine whether one of the thumb keys have been entered (blocks 43, 45, 57).

If the input character is a "T3" thumb key (block 43), the routine looks up the keycode from an internal table (not shown) for the control character corresponding to the appropriate finger key (block 44) and returns that keycode to the calling routine (block 67).

If the input character is a "T2" thumb key (block 45), the routine first checks if the "Up-Ctrl/En-Num" key was also entered (block 46). If so, the routine toggles a numeric character entry mode indicator (not shown) (block 47) and returns to the calling routine (block 67). If the "Tc/Ctrl" key was also entered (block 48), the routine accepts another input character from the keyboard (block 49) and looks up the keycode from an internal table (not shown) for the control character corresponding to the appropriate finger key (block 50). The routine returns that keycode to the calling routine (block 67). If the "Sp/Shf" key was also entered (block 51), the routine accepts another raw input character from the keyboard (block 52). The GET_CODE routine 21 then determines whether the "T2" thumb key was entered a second time (block 53) and if so, looks up the keycode for the capitalized alternative character corresponding to the appropriate finger key (block 54). Otherwise, if the "T2" thumb key was not entered a second time, the routine looks up the character keycode for the capitalized primary character corresponding to the appropriate finger key (block 55). In either case, the GET_CODE routine 21 returns the keycode to the calling routine (block 67).

If the original raw input character was a "T2"

thumb key and the "Up-Ctrl/En-Num", "Tc/Ctrl", or "Sp/Shf" keys were not entered, the GET_CODE routine 21 looks up the keycode from an internal table (not shown) for the alternate character corresponding to the appropriate finger key (block 56) and the routine returns; that keycode to the calling program (block 67).

If the raw input character is a "T1" thumb key (block 57), the GET_CODE routine 21 ascertains whether the "Tc/FN" key was also entered (block 58). If so, the routine gets another raw input character from the keyboard (block 59) and looks up the keycode from an internal table (not shown) for the function key character corresponding to the appropriate finger key (block 60). Otherwise, the routine looks up the keycode from an internal table (not shown) for the numeric character corresponding to the appropriate finger key (block 42). In either case, the routine returns the keycode to the calling routine (block 67).

If the T1, T2, or T3 thumb keys were not entered the GET_CODE routine 21 checks if the raw input character is the "Tc" key (block 61). If it is not, a primary character was entered and the routine looks up the keycode from an internal table (not shown) for the primary character corresponding to the appropriate finger key (block 62). Otherwise, the routine gets another raw input character from the keyboard (block 63) and determines whether the "T2" thumb key was entered a second time (block 64). If so, the routine looks up the keycode from an internal table (not shown) for the alternate terminal character corresponding to the appropriate finger key (block 65). Otherwise, the routine looks up the keycode from an internal table (not shown) for the primary character corresponding to the appropriate finger key (block 66). The routine returns the keycode to the calling routine (block 67).

Referring to FIG. 11, the GEN_CAP routine 22 is illustrated. The purpose of this routine is to generate a three word list of variations on the capitalization of the input word, prompt the user to select one of these words and return that word to the calling routine.

The GEN_CAP routine 22 begins by generating three variations of the input word stored in the buffer: first character capitalized, all characters capitalized and no characters capitalized (block 70). The routine then displays the first word in this list to the user (block 71) and calls the GET_CODE routine 21 to obtain the next keycode (block 72). If the input keycode is an "Up-Ctrl" key (block 73), the routine shifts the order of the list with the next word in the list at the beginning of the list (block 74). The routine again displays and prompts the user. Otherwise, the routine returns the selected word to the calling program (block 75).

For example, assume that a user enters an input character string comprising the characters "cat." The GEN_CAP routine 22 generates the three word list comprising "Cat, CAT, cat" (block 70). The routine displays the first word, "Cat", to the user (block 71) and gets the next input keycode (block 72). Assume that the user enters the "Up-Ctrl" key (block 73). The first word in the list becomes "CAT" (block 74) and this word is displayed to the user (block 71). The GEN_CAP routine 22 gets the next input keycode (block 72) and, assuming that it is not an "Up-Ctrl" key (block 73), the character string "CAT" is returned to the calling routine (block 75).

Referring to FIG. 12, a flow diagram for the AMB_RES routine 23 is illustrated. The AMB_RES routine 23 comprises the first level of ambiguity resolution logic. It seeks to find an exact and literal match

between the raw input string and its table of word pairings. The purpose of this routine is to find either an exact match for the input character string in the SH_DICT dictionary or to substitute a terminal character symbol for that string.

The routine begins by searching for an exact match between the character input string stored in the buffer against the set of entries stored in the SH_DICT dictionary (block 77). The SH_DICT dictionary comprises a set of word pairings. The structure and process for creating the SH_DICT dictionary is described below.

If an exact match between the input string and an index word in the SH_DICT dictionary is found (block 77), the routine checks if the input string matches a terminal character mnemonic corresponding to a frequently-used terminal character (block 78).

In the preferred embodiment, special consideration is given to selected punctuation marks that are frequently used in ordinary text. These terminal symbols include the period, comma, colon, semicolon, question mark, exclamation mark, backspace and carriage return. As set forth above, these punctuation symbols are enterable through the usual method for entering a character from the terminal character set. Remember that a terminal character can be entered by first entering the Tc key, followed by a terminal character finger key corresponding to the appropriate terminal character (see FIG. 7).

The usual method of entering terminal characters, especially those that are frequently used, causes an interruption in the natural flow of typing on the single-hand keyboard. This is primarily because the little finger is required to move away from its home row position thereby throwing off the timing of the user. Consequently, in the preferred embodiment, a

scheme is provided whereby selected frequently-used terminal characters can be entered by using a mnemonic key sequence, followed by the "Ar" key. The set of mnemonics is set forth in Table 2.

TERMINAL CHARACTER	INPUT CHARACTERS	MNEMONIC
. (period)	NE	PE
, (comma)	RO	CO
: (colon)	ROL	COL
; (semicolon)	OE	SE
? (question mark)	BD	QU
! (exclamation)	FE	XE
← (backspace)	EA	n/a
↵ (carriage return)	EE	n/a

Table 2

Referring to Table 2, three columns are shown. The first column lists the frequently-used terminal characters. The second column lists the primary input characters required to enter the mnemonic key sequence for the desired frequently-used symbol. The third column lists the mnemonic, where applicable, for the terminal symbol. The difference between the second and third columns is that the mnemonic column substitutes the appropriate alternate characters such that the mnemonic can serve as a memory device. For instance, "QU" is easier to remember than "BD".

An example illustrates this feature. Assume that a user wants to type a question mark using the mnemonic scheme. The user would type in "BD" (primary characters) followed by the "Ar" key. The AMB_RES routine 23 matches the input primary characters "BD" to

a known mnemonic (block 78) and substitutes for the characters "BD" that are stored in the buffer the question mark "?" (block 79). Note that the "Ar" key serves the special function of triggering the ambiguity resolution logic and causes it to substitute the desired terminal character for the two or three input characters comprising the mnemonic therein entered by the user.

If the input character string does not match a known mnemonic, the AMB_RES routine 23 calls the ENH_AMB_RES routine 25 and enters the second stage of ambiguity resolution logic (block 80). If the input character string does not match an entry in the SH_DICT dictionary, the AMB_RES routine 23 tries to apply word variation rules (block 82). It checks whether a possible stem of the word (root word) matches a word in the SH_DICT dictionary. This is for applying word prefix and suffix rules. Regardless of the result from the ENH_AMB_RES routine 25, the AMB_RES routine 23 returns to the calling routine (block 81).

B. Enhanced Ambiguity Resolution Logic

In its basic form, the ambiguity resolution logic resolves literal character string ambiguities. However, in the preferred embodiment, an additional level of ambiguity resolution logic is provided to enhance the basic functionality of the underlying system.

Referring to FIG. 13, the ENH_AMB_RES routine 25 is illustrated. This routine is the entry point from the first stage to the second stage and serves a three-fold purpose. First, it prompts the user with more word choices comprising the collection of words which have an identical index word. The collection is created in a dictionary generation process and is stored in the DICT dictionary.

Second, the routine presents to the user the various forms of verb tenses corresponding to the input word. For example, assume that the user has entered the primary characters "IO." The AMB_RES routine 23 resolves the character string ambiguity and prompts the user with the word "IS." The ENH_AMB_RES routine 25 displays the list "is, was, will be, has been, had been."

Third, the routine incorporates the ability to execute macros that are triggered by the input character string. For instance, the word "I" can be set up to fill in the name of the user, such as "Masakatsu Sugimoto" or to execute a function to display the time of day or set a stopwatch timer.

The ENH_AMB_RES routine 25 begins by creating a list of all words, if any, in the DICT dictionary that match the input string contained in the buffer (block 83). The structure and process for creating the DICT dictionary is described below. If no match is found and the list is an empty list (block 84), the routine merely returns to the calling routine (block 93). Otherwise, as with the AMB_RES routine 23, the ENH_AMB_RES routine 25 displays the first word in the list to the user (block 85) and calls the GET CODE routine 21 to obtain the next keycode (block 86). If the keycode matches the "Ar" key (block 87), the routine shifts the order of the list with the next word in the list at the beginning of the list (block 88). Otherwise, the routine determines if the input keycode is an "Up-Ctrl" key (block 89). If so, the GEN_CAP routine 22 is called (block 90) and the result is returned to the calling routine (block 93). If the input keycode is neither an "Ar" key nor an "Up-Ctrl" key, the first word in the list is accepted (block 92), the current character is placed in a lookahead buffer for further usage and returned to the calling routine

(block 93).

Note that the ENH_AMB_RES routine 25 introduces the most flexibility in terms of enhancing the functionality of the basic ambiguity resolution logic. The set of "words" in the DICT dictionary is of a general format. The first word is an index word. The second entry in the DICT dictionary can be any word, any list of words or a function value and it will be readily apparent to one skilled in the art that the level of functionality can be readily expanded upon. The set of enhanced features presented herein (tenses and macros) is merely illustrative and not meant to be an exclusive nor exhaustive list of enhancements to the basic ambiguity resolution logic.

A preferred embodiment of the ambiguity resolution logic has software written in Arity Prolog, a variant of the Prolog programming language. The definition of the Prolog programming language is detailed in W.F. Clocksin & C.S. Mellish, Programming in Prolog, (Cambridge 1984), herein incorporated by reference.

The software is preferably run on an IBM-compatible personal computer running the MS-DOS operating system 5.0 or higher. The Arity Prolog compiler and interpreter, Version 6.1, is loaded into the random access memory (RAM) of the personal computer. Preferably, the personal computer uses an Intel 80286-compatible microprocessor and is equipped with two megabytes or more of RAM.

C. Creation of Ambiguity Resolution Logic Dictionaries

The ambiguity resolution logic accesses a pair of data dictionaries, the DICT and SH_DICT dictionaries. The structure and process for creating these dictionaries is now described.

1. SH_DICT Dictionary

Referring to FIG. 14, the structure of the SH_DICT dictionary is illustrated. In the preferred embodiment, the dictionary comprises a set of word pairings organized into fourteen groups, one group for each character in the set of primary characters (A, B, D, E, F, I, J, K, L, M, N, O, R and P). The ambiguity resolution logic takes advantage of the first character of the input word when looking for a match in the SH_DICT dictionary. To speed up the search process, the dictionary is structured so that the first character of each input word also identifies the appropriate word group from within the dictionary.

The grouping label 100 identifies each of the fourteen word groups. The grouping label 100 comprises the primary character concatenated to the string "_SH_DICT." For example, the grouping label 100 corresponding to the primary character A is "A_SH_DICT."

Each word pair comprising an entry in the SH_DICT dictionary comprises an index word 101 and a matched word 102. The index word 101 corresponds to a character string that can be directly entered using the single-hand keyboard and using only primary characters. Stated differently, every word in the set of all index words 101 comprises only primary characters.

The matched word 102 is a correctly spelled word that substitutes alternate characters for those primary characters appearing in the index word 101 that cause the index word 101 to be ambiguous. On the other hand, not every word entered using primary characters is ambiguous. Thus, some of the word pair entries in the SH_DICT dictionary may have identical entries for index word 101 and matched word 102. Moreover, there may be several word pair entries in the dictionary that use the same index word 101.

Referring to FIG. 14, a diagram depicting an excerpt of the word grouping for the primary character A in the SH_DICT dictionary is shown. An example of how the ambiguity resolution logic operates is now discussed. Assume that the user enters the input character string "aorribe." The basic ambiguity resolution logic uses this character string as the index word 101 into the SH_DICT dictionary. Since the string beginning with the primary character "A", the ambiguity resolution logic only looks at the word grouping with the grouping label 100 that matches "A_SH_DICT." The ambiguity resolution logic then searches this grouping for a match between the character string "aorribe" and the set of index words 101 within that grouping. Since such an entry exists, the ambiguity resolution logic displays the matched word 102 that has the character string "ascribe."

Referring to FIG. 15, an overall block diagram of the process for creating the SH_DICT dictionary is shown. The purpose of this process is to establish a database of word pair entries organized into fourteen word groupings as set forth in the preceding discussion.

The process for creating the SH_DICT dictionary begins with the user creating an input file 105 comprising correctly spelled words that are to be recognized by the basic ambiguity resolution logic. The input file 105 is read by the CREATE_SH_DICT routine 106, which calls the SHKB_CONV routine 107 to transform the spelling of each word into a spelling comprising exclusively primary characters. The CREATE_SH_DICT routine 106 then writes the resultant word pair entry to the SH_DICT dictionary 108.

Referring to FIG. 16, a flow diagram for the CREATE_SH_DICT routine 106 is illustrated. The purpose of this routine is to transform each word occurring in

the input file 105 into the set of word pair entry groupings, one grouping for each primary character, as contained in the SH_DICT dictionary 108. Each word occurring in the input file 105 is converted into a spelling comprising only primary characters.

The routine begins by opening the input file 105 (block 110) containing the words to be used by the basic ambiguity resolution logic. The routine then reads the first word in the input file 105 (block 111) and calls the SHKB_CONV 107 to convert the spelling of the input word into exclusively primary characters (block 112). The SHKB_CONV routine 107 stores the converted input word in a temporary buffer. The routine then creates a word pair entry in the SH_DICT dictionary 108 comprising a grouping label 100 associated with a word pair comprising the index word 101 and the matched word 102. Recall that the indexed word 101 is made up of primary characters exclusively and the matched word 102 corresponds to the input word as read in from the input file 105 (block 113). If the routine has reached the end of the input file 105 (block 114), no words remain in the input file 105 and the routine terminates (block 116). Otherwise, the CREATE_SH_DICT routine 106 gets the next word in the input file 105 (block 115) and the conversion process continues anew.

Referring to FIG. 17, a flow diagram for the SHKB_CONV routine 107 is illustrated. The purpose of this routine is to convert a single input word into an output word stored in a temporary buffer that is spelled only with primary characters.

The SHKB_CONV routine 107 begins by accepting an input word and setting an internal pointer to the first character of that word (block 120). The routine then parses the input word one character at a time and converts each character into a primary character. If

the current character is already a primary character (blocks 123, 125, 127, 133, 135, 139, 141, 143, 147, 149, 157, 159, 161 and 171), the character is mapped to a lower case representation of that primary character (blocks 124, 126, 128, 134, 136, 140, 142, 144, 148, 150, 158, 160, 162 and 172). Otherwise, the current input character is an alternate character (block 121, 129, 131, 137, 145, 151, 153, 155, 163, 165, 167 and 169) and is mapped to the lower case representation of the primary character assigned thereto (block 122, 130, 132, 138, 146, 152, 154, 156, 164, 166, 168 and 170).

The SHKB_CONV routine 107 also handles two special cases. First, if the current input character is a space (block 173), the character is mapped to a lower case B (block 174). Second, if the input character is a period (block 175), the character is also mapped to a lower case B (block 176). Otherwise, if the current input character is neither a primary or alternate character, nor a space or period, the character is mapped to a question mark (block 177).

Regardless of the result of the mapping process, the routine concatenates the mapped character to the temporary buffer (block 178) and increments the internal pointer to the next character of the input word (block 179). The routine assumes that every input word is null terminated, therefore, if the next input character is a null character (block 180), the end of the word has been reached and routine returns the converted word in the temporary buffer (block 181). Otherwise, the routine begins the parsing process anew.

2. DICT Dictionary

The purpose of the DICT dictionary is to provide a database that supplies an additional level of functionality to the basic ambiguity resolution logic. Referring to FIG. 18a, a diagram depicting an excerpt

of the DICT dictionary is illustrated. It comprises a set of entries which include an index word 184 and a list 185. The index word 184 is a correctly spelled word. As a result, the DICT dictionary is only used by the enhanced ambiguity resolution logic after the initial input word ambiguity has been resolved. The list 185 comprises one or more words corresponding to alternate spellings of the index word 184, that is, alternate characters are substituted for one or more of the characters comprising the index word 184.

Referring to FIG. 18b, a diagram depicting an excerpt of an alternate embodiment of the DICT dictionary is shown. Rather than using the approach taken in the preferred embodiment, the alternate embodiment shown "flattens out" the list 185 by creating word-pair entries structured similarly to the SH_DICT dictionary 108 (see FIG. 14). Consequently, multiple entries may be present in the DICT dictionary which use the same index word 184 yet have a different matched word 186, each such matched word 186 comprising one of the words found in the list 185.

Referring to FIG. 19, an overall block diagram of the process for creating the DICT dictionary is illustrated. The CREATE_DICT routine 189 reads the set of word pair entries stored in the SH_DICT dictionary 108. It then iteratively calls the CHECK_MULT routine 190, one call for each primary character. In turn, the CHECK_MULT routine 190 records each entry from the SH DICT dictionary 108 in a temporary database called the EXIST database 191. The CHECK_MULT routine 190 checks if multiple index words occur in the SH_DICT dictionary 108 and records every multiple occurrence in another temporary database, the MULTIPLE database 192. After the SH DICT dictionary 108 has been processed, the CREATE_DICT routine 189 calls the PROC_MULT routine 193. That routine reads each entry in the MULTIPLE

database 192 and consolidates each multiple entry into a single entry that it stores into the DICT dictionary 194.

Referring to FIG. 20, a flow diagram for the CREATE_DICT routine 189 is illustrated. The purpose of this routine is to process each of the groupings of word pair entries stored in the SH_DICT dictionary 108 for duplicate entries.

The routine begins by opening the SH_DICT dictionary 108 for reading. The routine then iteratively calls the CHECK_MULT routine 190, one call for each primary character (block 198). The CHECK_MULT routine 190 analyzes the word grouping for the current primary character (block 199) and repeats the process until the last primary character has been processed (block 200). Once completed, the CREATE_DICT routine 189 calls the PROC_MULT routine 193 to process any multiple entries detected in the SH_DICT dictionary 108 (block 201) and terminates (block 202).

Referring to FIG. 21, a flow diagram for the CHECK_MULT routine 190 is illustrated. A purpose of this routine is to detect duplicate entries in the SH_DICT dictionary 108 and to record them for processing by the PROC_MULT routine 193.

The CHECK_MULT routine 190 begins by retrieving a word pair entry from the .SH_DICT dictionary 108 (block 205). The routine uses the variable CODE to store the first (index) word comprising the word pair and the variable ENTRY to store the second (matched) word in the word pair. The word pair entry is recorded in a temporary database, the EXIST database 191 (block 206).

Next, the routine performs a look up to the EXIST database 191 using the current index word as stored in the variable CODE to see if a duplicate entry exists in the EXIST database 191 (block 207); if a duplicate entry is detected, the CHECK_MULT routine 190 records

that entry in another temporary database, the MULTIPLE database 192. The routine uses the variable MULTIPLE to store the duplicate matched word. The process of retrieving an entry from the SH_DICT dictionary 108 and searching the EXIST database 191 for duplicate entries continues until the last entry for the current primary character is encountered (block 209), whereupon the routine returns (block 210).

Referring to FIG. 22, a flow diagram for the PROC MULT routine 193 is illustrated. The purpose of this routine is to process each duplicate entry as recorded in the MULTIPLE database 192 and to write a consolidated entry into the DICT dictionary 194.

The PROC_MULT routine 193 begins by retrieving an entry from the MULTIPLE database 192 (block 213). It then concatenates the second entry from the word pair to the variable LIST (block 214). The routine continues to process all duplicate entries indexed under the same index word (as stored in the variable CODE) until the last entry has been encountered (block 215).

After all entries for the same index word are processed, the PROC_MULT routine 193 writes an entry into the DICT dictionary 194 comprising the index word and the list of duplicate entries corresponding to that index code word (block 216). When the last entry in the MULTIPLE database 192 has been processed (block 217), the routine returns (block 218).

III. Software Emulator and Hardware Embodiment

One embodiment of a single-hand keyboard comprises a software emulator operating on a conventional personal computer to transform a conventional full keyboard into a single-hand keyboard. Another embodiment comprises a dedicated single-hand keyboard

similar to a personal data assistant.

A. Software Emulator

One embodiment of the single-hand keyboard is an emulator which uses a personal computer program with a computer program to map the keys on a full, conventional two-handed keyboard for single handed use. This enables a user to transform a full keyboard into either the right-handed single-hand keyboard 5 or left-handed single-hand keyboard 7. There are several applications of the emulator including use as a test bed, for training purposes, or use to enable single-handed typing to free the other hand for operations such as holding a telephone handset.

The software emulator utilizes a subset of the keys found on a conventional full keyboard. Referring to FIG. 23a, the key layout of a conventional keyboard 300 indicating in heavy lines the keys 301 for a right-handed single-hand keyboard 5' is shown. Similarly, referring to FIG. 23b, the key layout of a conventional keyboard 302 indicating in heavy lines the keys 303 for a left-handed single-hand keyboard 7' is shown.

Referring to FIG. 24, a flow diagram for a SHKB_MAP computer program routine 220 is illustrated. The purpose of this routine is to map the raw signals (representative of the keys that are entered) output from a full keyboard and keyboard controller using a conventional personal computer. The mapping is done in a manner that emulates a single-hand keyboard. Although the SHKB_MAP routine 220 as depicted in FIG. 24 is for a right-handed single-hand keyboard 5', it will be obvious to one skilled in the art as to how to modify this routine to function as an emulator for a left-handed single-hand keyboard 7'.

The SHKB_MAP routine 220 is designed to operate as

a driver program. Consequently, it interfaces directly to the hardware and would be transparent to the ambiguity resolution logic. The SHKB_MAP routine 220 begins by getting an input word from the keyboard (block 221) and setting an internal pointer to the first character in that input word (block 222). The routine then parses that raw input character (blocks 223, 225, 227, 229, 231, 233, 235, 237, 239, 241, 243, 245, 247 and 249) and maps it to one of the 14 primary characters recognized by the single-hand keyboard (blocks 224, 226, 228, 230, 232, 234, 236, 238, 240, 242, 244, 246, 248 and 250). If the raw input character is not recognized, the SHKB_MAP routine 220 maps that character to the character B (block 251).

The routine concatenates the mapped character to a temporary input buffer (block 252) and increments the internal pointer to the next character in the input word (block 253). The routine assumes that the input word is null terminated and when the null character is detected (block 254), the routine returns the mapped input word to the ambiguity resolution logic and terminates (block 255).

B. Dedicated Single-Hand Keyboard

FIG. 25 discloses a keyboard entry system which includes dedicated single-hand keyboard 5' and display screen 277 that can be held in the palm of a person's hand. The keyboard entry system 275 communicates via electronic interface 278 to personal computer 279.

The keyboard 5' comprises finger keys and thumb keys 281 that are essentially the same as that of keyboard 5 of FIG. 2 except as discussed below. An electronic interface (not shown) located inside the Cabinet of dedicated single-hand keyboard 5' is a conventional RS-232C or RS422 serial connection, a dedicated bus interface, a wireless interface, a PCMCIA

interface, a standard 5-wire DIN keyboard interface, or a row/column interface to interface with personal computer 279. The foregoing list of interface devices is not meant as a limitation and other interfaces could be employed within the spirit of the subject invention.

As the keys are entered on keyboard 5', as discussed with reference to keyboard 5, the characters that are generated are displayed on the display screen 277 in a conventional manner well known in the art of laptop computers. This enables the user to see the sequence of characters that are being generated as a sequence of keys is entered. This enables the user to use the "CTVi" key to change characters from a primary to a secondary character assigned to a multiple-character key that is being entered or, after a sequence of keys have been entered, to enter the "Ar" key to change the sequence of characters using the ambiguity resolver while viewing the sequence of characters on the display screen 277. The resultant sequence of generated characters can then be sent to computer 279.

FIG. 26 depicts a schematic and block diagram of a single-hand keyboard entry system 275. The keypad 276, which includes the keys of one of the single-hand keyboards described above or a conventional two-hand keyboard with character mapping to a single-hand keyboard as shown in FIG. 24. The keys are unrelated through key switches (not shown), to dedicated keyboard controller 285 which transmits a scan code signal to the microprocessor 286. The microprocessor 286, under computer program control, carries out the character generation and implements ambiguity resolution as described above and transmits the words or sequences of characters in their final, correctly spelled and in unambiguous form to the personal computer 279 (FIG. 25).

The dedicated single-hand keyboard system 275 is self-powered by battery pack 287 which is controlled by on/off switch 288 and supplies power for the entire keyboard system 275. A reset button 289 is provided to reset the microprocessor 286 in the event of a computer program failure or user intervention. The microprocessor 286 echoes the keys entered by the user on the display screen 277. In the preferred embodiment, the display screen 277 is a 40 column by 20 row liquid crystal diode display.

Communication to the personal computer 279 is via the electronic interface 278 which is connected to the microprocessor 286. An optional interface is the PCMCIA interface 290 containing a two megabyte (MB) memory card. By way of example, the microprocessor 286 is equipped with two external memory stores, the random access memory (RAM) 291 and the read-only memory (ROM) 292; both the RAM 291 and the ROM 292 have 2MB capacities and the ROM 292 is dedicated to storing the operating system for the dedicated single-hand keyboard.

In one embodiment of the dedicated single-hand keyboard system 275, the microprocessor 286 is an Intel 80286-compatible microprocessor with a clock speed of 8 MHz or faster. Additionally, the preferred operating system for the microprocessor is MS-DOS Version 5.0 or higher. Such an embodiment of the invention could be incorporated into a word processing system or as a PDA. The character generator and the ambiguity resolver can be incorporated into or used with a word processor. A the keyboard and switches of FIG. 26 can be in one unit and the rest of the system can be part of a computer where the display is part of the keyboard unit or in the computer.

While the invention has been particularly shown and described with reference to the preferred

embodiments thereof, it will be understood by those skilled in the art that the foregoing and other changes in form and detail may be made therein without departing from the spirit and scope of the invention.

CLAIMS:

1. A data entry system comprising: keys each assigned a character, the keys comprising multiple character keys, each multiple character key being assigned multiple characters;

a character generator for generating upon entry of a sequence of the keys, a sequence of the characters comprising a character assigned to each of the sequence of the keys, there being an ambiguity among the multiple characters assigned to a multiple character key that is entered as to the correct assigned character that should be included in the sequence of characters; and

a multiple character resolver with word variation logic responsive to the sequence of keys that are entered for resolving the ambiguity.

2. A data entry system according to claim 1, wherein the character generator comprises a processor.

3. A data entry system according to claim 2, wherein the processor comprises a programmed computer.

4. A data entry system according to any one of claims 1 to 3, wherein the resolver comprises a table comprising plural sets of the characters from which one of the sets of characters is selected to replace the sequence of characters.

5. A data entry system according to claim 4, comprising a selector for selecting a set of characters from the table based on at least some of the keys that are entered.

6. A data entry system according to claim 4 or 5, wherein each of the sequence of characters is a word.

7. A data entry system according to any one of the claims 4 to 6, comprising a memory and wherein the resolver comprises:

a dictionary stored in the memory, the

dictionary comprising a list of words, each word comprising a sequence of the characters; and

means for selecting one of the words in the dictionary based on the sequence of the characters assigned to the sequence of keys that are entered.

8. A data entry system according to any one of the claim 1 to 7, comprising a display for displaying to the user the sequence of the characters assigned to the entered keys.

9. A date entry system according to claim 8, comprising a buffer for storing the sequence of characters for the display.

10. A data entry system according to any one of the claims 7 to 9, wherein the multiple character keys are each assigned a primary character that is normally generated by entry of the corresponding key and an alternate character and wherein the dictionary comprises:

first and second sets of words;

the first set comprising a set of words comprising the primary characters; and

the second set comprising a set of words comprising both the primary characters and the alternate characters, each word in the first set corresponding to one of the words in the second set and each of the words in the second set comprising only said primary characters and alternate characters that are assigned to the same keys as the corresponding word in the first set.

11. A data entry system according to any one of the claims 7 to 9, wherein the dictionary comprises:

first and second sets of words;

the first set comprising a set of words comprising both the primary characters and the alternate characters; and

the second set comprising a set of word

lists, each word in the first set corresponding to one of the word lists in the second set and each of the words in the word list in the second set comprising only said primary characters and alternate characters that are assigned to the same keys as the primary characters and alternate characters for the corresponding word in the first set.

12. A data entry system according to any one of the preceding claims, comprising means for capitalizing one or more of the characters comprising the sequence of characters.

13. A data entry system according to any one of the preceding claims, comprising means for inserting a punctuation mark in the sequence of characters.

14. A data entry system according to any one of the preceding claims, comprising means for automatically generating a grammatically correct prefix or suffix to the sequence of characters.

15. A data entry system according to any one of the preceding claims, comprising means for automatically generating a grammatically correct verb tense variation of the sequence of characters.

16. A data entry system according to any one of the preceding claims, wherein the character generator comprises means for automatically executing a user-definable macro in response to the sequence of keys for generation of the sequence of characters.

17. A data entry system according to any one of the preceding claims, into which words can be input, the keyboard comprising a plurality of keys, each of which having a primary character and at least one of these keys having a plurality of alternate characters, an input word being such that it may be ambiguously spelt;

logic for accepting an input word from the keyboard;

logic for matching an input word to a list comprising one or more correctly spelt words; and

a display for displaying one or more correctly spelled words to a user in the event that an input word is ambiguously spelt.

18. A data entry system for resolving ambiguity in a word input thereto comprising:

a keyboard having a plurality of keys suitable for keying in words, each of these keys having a primary character and at least one of these keys having a plurality of alternate characters, an input word being such that it may be ambiguously spelt;

a character buffer for storing a plurality of character keycodes;

logic for obtaining a character keycode from the plurality of keys;

logic for returning a word corresponding to the contents of the character buffer when the character keycode corresponds to a carriage return key;

logic for truncating the contents of the character buffer by one character when the character keycode corresponds to a backspace key;

logic for setting the contents of the character buffer to a null string when the character keycode corresponds to an escape key;

logic for performing ambiguity resolution with word variation logic on the contents of the character buffer when the character keycode corresponds to an ambiguity resolution key;

logic for storing the character keycode in the character buffer when the character keycode does not correspond to a carriage return key or a backspace key or in escape key or an ambiguity resolution key;

logic for matching an input word to a list comprising one or more correctly spelt words; and

a display for displaying one or more

correctly spelled words to a user in the event that an input word is ambiguously spelt.

19. A data entry system according to claim 17 or 18, wherein the keyboard comprises a plurality of thumb keys and the input word either comprises one or more input characters matching one of the primary characters or comprises one or more input characters matching one of the alternate characters, the alternate characters enterable by simultaneously pressing a key and a thumb key.

20. A data entry system according to claim 17, 18 or 19, comprising:

a dictionary having a primary set and an alternate set,

the primary set comprising a plurality of index words, each index word comprising one or more primary characters and

the alternate set comprising a plurality of word lists, each word list comprising one or more words and corresponding to an index word, each word comprising one or more primary or alternate characters.

21. A data entry system for resolving word ambiguities comprising:

a plurality of keys, at least some of the keys being assigned at least one character and at least one key being assigned a plurality of characters;

a character buffer for storing a plurality of character keycodes;

logic for obtaining a character keycode from the plurality of keys;

logic for returning a word corresponding to the contents of the character buffer when the character keycode corresponds to a carriage return key;

logic for truncating the contents of the character buffer by one character when the character keycode corresponds to a backspace key;

logic for setting the contents of the character buffer to a null string when the character keycode corresponds to an escape key;

logic for performing ambiguity resolution with word variation logic on the contents of the character buffer when the character keycode corresponds to an ambiguity resolution key; and

logic for storing the character keycode in the character buffer when the character keycode does not correspond to a carriage return key or a backspace key or in escape key or an ambiguity resolution key.

22. A data entry system according to claim 21, comprising:

a memory store;

a dictionary stored in the memory store;

logic for performing ambiguity resolution comprising:

logic for matching the contents of the character buffer to a word in the dictionary,

logic for processing word variations when no match is found and

logic for substituting the contents of the character buffer with a terminal symbol when a match is found and the word corresponds to a predetermined mnemonic;

logic for performing enhanced ambiguity resolution logic when no match is found or when a match is found but the word does not correspond to a predetermined mnemonic;

a display for displaying the word matched in the dictionary when a match is found; and

logic for prompting the user to select the word.

23. A data entry system according to any one of claims 1 to 17, wherein the character generator has a keyboard comprising keys for entry of characters, each said key assigned a character and each of at least some

of said keys being multiple character keys assigned multiple characters, the character generator being operable to generate one of the characters assigned to each of a sequence of the character keys that are entered on the keyboard, there being an ambiguity as to the correct character to be generated when any said multiple character key is entered in the sequence of keys, the character generator comprising an ambiguity resolver for operating on a sequence of character keys that have been entered, to resolve the correct character for any of the multiple character keys of said sequence of character keys.

The Patent Office

-51-

Application No: GB 9813753.2
Claims searched: 1-23

Examiner: Gary Williams
Date of search: 5 September 1998

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.P): G4H: HKK,HKH

Int Cl (Ed.6): G06F: 3/023; H03M: 11/04,11/08; H04M: 1/00,3/62,9/00,11/06

Other:

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X,P	GB 2283598 A	(IBM) See Figs.1&2, page 2 lines 11-24	1,2,4,6-8,17,23
X	GB 2266797 A	(NOKIA) See Fig.2, page 10 line 32 - page 11 line 4	1,8
X	US 4650927	(IBM) See col.1 line 60 - col.2 line 9	1

X Document indicating lack of novelty or inventive step
Y Document indicating lack of inventive step if combined with one or more other documents of same category.
& Member of the same patent family
A Document indicating technological background and/or state of the art.
P Document published on or after the declared priority date but before the filing date of this invention.
E Patent document published on or after, but with priority date earlier than, the filing date of this application.